The Dissertation Committee for Yongseok Cheon

certifies that this is the approved version of the following dissertation:

Multilevel Circuit Partitioning for Computer-Aided VLSI Design

Committee:

Aloysius K. Mok, Supervisor

Martin D. F. Wong, Co-Supervisor

Donald S. Fussell

Mohamed G. Gouda

David Z. Pan

Multilevel Circuit Partitioning for Computer-Aided VLSI Design

by

Yongseok Cheon, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2004

In memory of my father

Acknowledgments

First, I would like to sincerely thank my supervising professor, Martin D. F. Wong for his invaluable support and guidance. His encouragement and insights on various issues throughout my Ph.D. study at UT were really priceless.

I'm greatly grateful to professors Al Mok, Don Fussell, Mohamed Gouda, and David Pan, for serving on my committee, showing interests in my work and providing inspiring comments on my dissertation. I also appreciate help from the department's administrative staffs, and especially I thank Gloria Ramirez in the graduate office for her assistance.

I enjoyed every moment shared with all the members of VLSI CAD group, including Minghorng Lai, Hung-Ming Chen, Xiaoping Tang, Li-Da Huang, Muzhou Shao, Hua Xiang, Gang Xu, and Xia Chen. Special thanks goes to Seokjin Lee; the conversations with him were always enjoyable and inspiring. I also would like to thank my wonderful friends in CS department, Chan-Gun Lee, Honguk Woo, Dongyoung Lee and many others.

My sincerest gratitude goes to my mother, my wife's parents, and my sisters for their bottomless love and support. I owe special thanks to my 'beloved-forever' wife, Eunhee Jang, and my most precious three daughters, Sohui, Rachel, and Hailey.

I will always be there for them.

Yongseok Cheon

The University of Texas at Austin

August 2004

Multilevel Circuit Partitioning for Computer-Aided VLSI Design

Publication No. _____

Yongseok Cheon, Ph.D.

The University of Texas at Austin, 2004

Supervisors: Martin D. F. Wong and Aloysius K. Mok

As designs become massively interconnect-dominated and present unmanageable instance complexities, circuit partitioning is recognized as a critical optimization problem in computer-aided VLSI design automation; the feasibility as well as the quality of the automatic placement and routing procedures heavily depends on the quality of partitioning solutions. In this dissertation, the weakness and the limitation of current partitioning techniques are identified, and new partitioning algorithms are suggested. Not only we define new metrics for additional qualities of partitioning solutions, but we also present novel partitioning techniques for large scale designs.

First, we present a new multilevel circuit partitioning algorithm which is guided by design hierarchies. In addition to a flat netlist hypergraph, a logical design hierarchy is used as a guidance for multilevel partitioning. Using Rent's exponent as a quality indicator for physical connectivities of the hierarchical elements, multilevel clustering scopes are guided (and dynamically restructured) by strongly connected

hierarchical elements in the design hierarchy. By exploiting the design hierarchy, our algorithm produces higher quality solutions than conventional multilevel partitioners and the solutions are also significantly more stable over the multiple runs.

Second, we define stability as an additional quality measure for partitioning solutions, and a new stable multiway partitioning algorithm is presented. Given a previous partitioning result P^* on an original netlist hypergraph H^* and a partially modified netlist hypergraph H, a new cost function with similarity factor is defined to produce a new partition P on H which is similar to the original partition P^*. We also present a multilevel version of the algorithm with multilevel restricted coarsening. The proposed partitioner is especially beneficial to engineering change order (ECO) applications, where partial modifications of a netlist are handled by the incremental methodology in a design iteration cycle.

Third, we propose a new multilevel multiway partitioning approach which is aware of the resource utilization distribution, assuming the resource utilization per partitioned block is proportional to the logic occupation rate and required interconnection. A new quality, crowdedness, is defined as a virtual complexity metric where the physical size and the local connectivity of a partitioned block are combined as the weighted sum. Unlike conventional partitioners focusing on the overall interconnection minimization, which have wide variances in the crowdedness(equivalently, resource utilization) distribution, the proposed algorithm produces near-optimal solutions in terms of crowdedness distribution while overall interconnection quality is also improved.

Lastly, we present an improved version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree constructed

during the coarsening phase. Though the multilevel partitioning paradigm is relatively robust, it has limited flexibility; while the partitioning solution is refined over multiple levels, the problem instance at each level totally relies on the construction of the multilevel clustering tree. In the proposed version of multilevel partitioner, the multilevel uncoarsening phase involves multi-rate unclustering, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels. The superiority of the suggested multilevel partitioner is justified by the experimental results.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Circuit Partitioning Problem

Given a set of cells (circuit elements) and the nets that connect these cells, the circuit partitioning problem is to partition these cells into several disjoint blocks (subcircuits) of specified sizes such that the number of interconnections between blocks is minimized. The circuits are typically represented by hypergraphs [27, 8] as illustrated in Figure 1.1. Circuit partitioning is a critical optimization problem in many areas of VLSI design automation because the partitioning solutions have a great impact on automatic placement and routing procedures, especially in large scale design procedures, where tens of millions of cells are handled. Millions of the cells cannot be handled in a flat mode any more due to the limitation of computation power and memory space, and concurrent engineering of the individual subcircuits can shorten design turnaround time. As a result, a circuit needs to be partitioned into several blocks where the massive amount of data is broken into manageable sizes. Also partitioning techniques are often embedded in the large scale placement

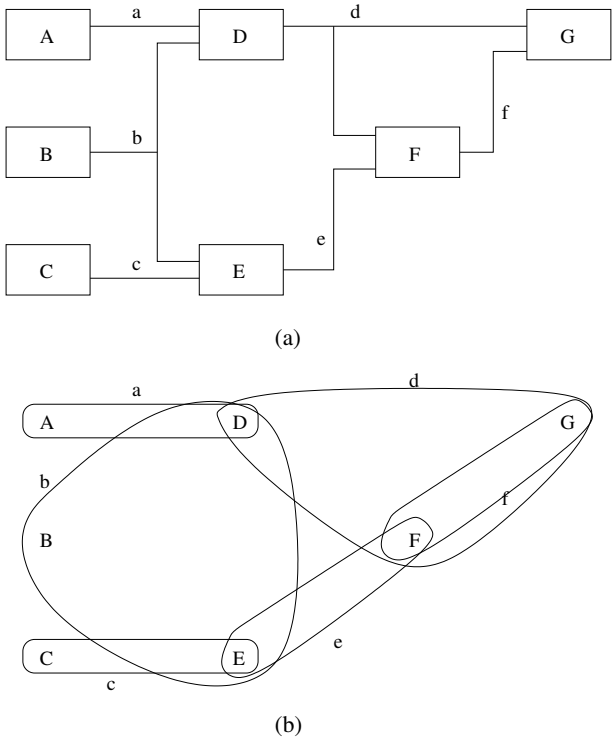


Figure 1.1: A circuit modeled as a hypergraph. (a) Original circuit consisting of seven cells and six nets, (b) Hypergraph representation with vertices and hyperedges.

procedure to determine the optimal global positions [12, 9, 38, 44, 29].

The attempts to solve this NP-hard problem [34] have concentrated on finding heuristic algorithms which yield near-optimal solution in polynomial time [58, 14]. Some of the best known approaches include the move-based iterative improvement methods such as Kernighan-Lin (KL), Fiduccia-Mattheyses (FM) algorithms and their variations, which are called iterative improvement partitioning (IIP) techniques [33, 43, 59, 57, 11]. Unlike simulated annealing (SA) approaches [45, 37] which also are move-based, iterative improvement algorithms are based on the greedy strategy. They start with some feasible solution and iteratively move to the best possible neighboring solution. The process is successively performed until

Procedure `IIP`

Input: a given netlist hypergraph hypergraph $H = (\mathcal{C}, \mathcal{N})$
K = number of blocks
Output: K-way partition $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$

1. Construct a balanced random partition P
2. **repeat**
3. Set the initial partition $P_i := P$
4. **while** there exists a movable free cell **do**
5. Select a free cell C with the highest gain (cost reduction)
($\mathcal{A}_i$ = source block, $\mathcal{A}_j$ = destination block)
6. Assign cell C to block $\mathcal{A}_j$
7. Lock cell C
8. Update gains of the affected neighboring free cells
9. **end while**
10. $P :=$ the best partition with the lowest cost among
the partitions produced in above while loop
11. **until** there is no improvement in the cost of partition

Figure 1.2: Basic structure of iterative improvement partitioning (IIP) algorithm. An entire execution of the while loop is called a pass, hence one run of IIP algorithm with one starting partition consists of a number of passes.

the algorithm reaches a local minimum, i.e., a solution for which all neighboring solutions have greater costs. (The brief outline of general IIP partitioning algorithm is shown in Figure 1.2). However, the IIP strategies discussed in this dissertation all rely on extended neighborhood structures which effectively allow hill-climbing out of local minima. While these IIP algorithms have been extended in a number of ways [46, 56, 31, 30, 24, 13, 66, 15], some new strategies using combinatorial formulations [65, 53] and geometric representations [36, 5, 16] have been explored, as well.

Clustering is also a sort of partitioning in the sense that it also divide the cells into disjoint clusters based on the connectivity. However, clustering is distinguished by that the number of desired clusters are usually not predefined and the sizes of

clusters are relatively small. Clustering-based approaches are often done in bottom-up fashions, and used as preprocessing techniques to reduce the problem size before the top-down partitioning heuristics are applied. This (bi-level) two-phase approach has been applied with many clustering algorithms [3, 26, 35].

Partitioning heuristics also have an impact on system performance as designs become interconnect-dominated. In current submicron designs, wire delays tend to dominate gate delays [6]. Since the number of signals which pass between the components corresponds to the interactions between the design subproblems, the differences between on-chip and off-chip signal delays and the pin-limited nature of large chips makes it desirable to minimize the number of signals traveling off a given chip. A typical application in this category is the multi-FPGA systems, where each FPGA is given a fixed available size for logics and a limited number of pins for connection with other FPGA's [10, 53, 67, 49, 22, 48]. Partitioning heuristics affect the layout area, as well. Wires between subcircuits at high levels of the hierarchy will tend to be longer than wires between subcircuits at lower levels, and total wirelength is proportional to layout area due to wire spacing design rules. In this sense, the partitioning technique is natural for this application; if the layout area is divided into a dense uniform grid, total wirelength can be expressed in "grid" units or equivalently as the sum over all gridlines of the number of wires crossing each gridline. This view can also improve routability since it suggests reducing the wire congestion in any given layout region.

Multiway partitioning can be implemented in two different methods; one is applying the FM bipartitioning algorithm recursively (recursive bipartitioning, RBP) [33, 43], and the other is a direct K-way partitioning, an extension of the FM

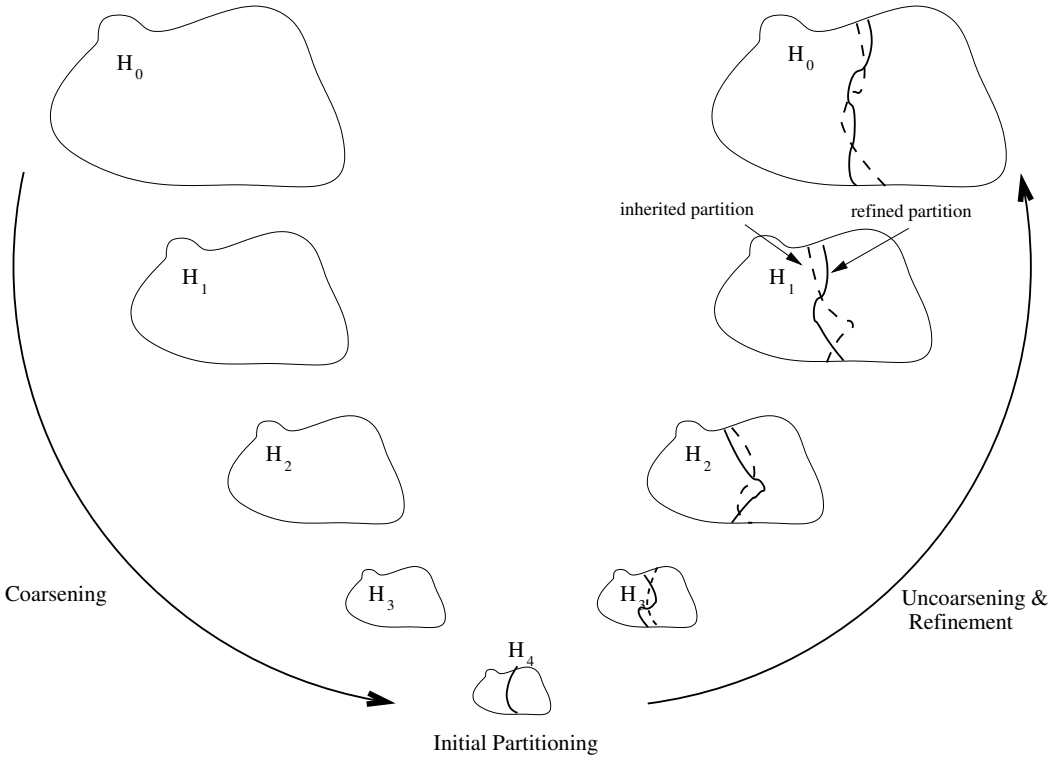


Figure 1.3: Multilevel hypergraph bi-partitioning which consists of coarsening, initial partitioning, and uncoarsening and refinement phases. H_i is the next level coarser hypergraph of H_{i-1}.

algorithm to multiway partitioning [57, 55]. In general, the recursive bipartitioning technique yields higher quality solutions than direct K-way partitioning. Though direct multiway partitioning tends to produce relatively worse quality solutions and requires more amount of memory than RBP technique, it shows faster runtime and higher balance management capability. Maintaining a global view over entire blocks enables the direct multiway partitioning to be suitable for quick partitioning refinement or incremental partitioning in engineering change order (ECO) situations, where partial modifications of a circuit are handled by the incremental methodology in the design iteration cycle.

Recently, the new multilevel partitioning paradigm has been introduced in order to improve partitioning results of iterative improvement approaches especially for bigger designs [2, 39, 41, 20]. The multilevel partitioning has been shown to be the most effective approach to producing excellent partitioning quality in a comparatively short run time. Multilevel partitioning consists of three phases: multilevel coarsening, initial partitioning at the coarsest level, and multilevel uncoarsening with FM refinement (See Figure 1.3). During the coarsening phase, the problem size is gradually reduced over the levels while capturing strong connectivities in the circuit netlist. At the coarsest level, a relatively high quality initial partitioning solution is quickly obtained. Then, the current partitioning solution is successively propagated to lower levels, where the partitioning solution on the bigger problem keeps improved by FM refinement. A deeper analysis of the multilevel partitioning is found in Chapter 5.

1.2 Dissertation Overview

In this dissertation, the weakness and the limitation of current partitioning techniques are identified, and new partitioning algorithms are suggested for each of them. Not only we define new metrics for additional qualities of partitioning solutions, but we also present novel partitioning techniques for large scale designs.

- **Lack of logical grouping information.** In most practical cases, a design is provided as a hierarchy based on functional and/or spatial decomposition determined by logic designers. However, this logical grouping information has been totally ignored in the physical design domain even though such hierarchical decomposition of the design also implies the physical connectivity [20, 21].

Chapter 2 presents a new multilevel circuit partitioning algorithm (`dhml`) which is guided by design hierarchy. In addition to flat netlist hypergraph, we use user design hierarchy as a hint for partitioning. This design hierarchy already has some implications on connectivity between logical blocks in the design. Using design hierarchy in partitioning is nontrivial since the hierarchical elements in design hierarchy do not necessarily have strong internal connectivity; hence we need to determine whether it is preferable to break up or preserve the hierarchical elements. In order to identify and select the hierarchical elements with strong connectivity, their Rent exponents are used. Then, the selected hierarchical elements serve as effective clustering scopes during the multilevel coarsening phase. The scopes are dynamically updated (enlarged) while building up a clustering tree so that the clustering tree resembles the densely connected portions of the design hierarchy.

- **Instability of the partitioning solutions.** Iterative improvement partitioning approaches heavily rely on multiple runs in order to obtain a best solution among them. The different runs with different initial partitions lead to a wide spectrum of solutions, from which the best quality solution is picked. Even though this instability gives us opportunities to find higher quality solutions through the multiple runs, it ironically yields a critical weakness in incremental applications [19, 17]; very minor changes to a circuit netlist can produce radically different results, hence it is difficult to achieve timing closure in the following block-level incremental placement and routing procedures.

  In Chapter 3, we propose a new *stable* multiway partitioning algorithm, where stability is defined as an additional quality of a partitioning solution. The sta-

bility of a partitioning algorithm is an important criterion for a partitioning based placement to achieve timing closure through the repetition of the placement procedure [63]. Given a previous partitioning result P^* on an original netlist hypergraph H^* and a partially modified netlist hypergraph H, a new cost function with similarity factor is defined to produce a new partition P on H which is *similar* to the original partition P^*. The proposed algorithm is the first approach that quantifies the degree of similarity of a current partition to the original partition using similarity cost. Our goal is to build a new partition in a relatively short run time, whose cut quality is not much degraded from that of the original partition P^* while it preserves as much of the previous groupings in P^* as possible. We also present a multilevel version of the algorithm with restricted coarsening, and it shows a dramatic speed-up without the sacrifice of stability and cut quality. The proposed partitioner is especially beneficial to engineering change order (ECO) applications, where partial modifications of a netlist are handled by the incremental methodology in a design iteration cycle. Our approach helps ECO placers maximize the incremental capability since the portions to be re-placed are minimized.

- **Lack of control over the interconnection distribution.** In general, partitioning algorithms focus on the overall interconnection minimization among the partitioned blocks. Since the distribution of the interconnection over the individual blocks are ignored in partitioning, the imbalance of the interconnection distribution is significantly severe even though the block sizes are still the feasibility condition [18, 60]. Assuming the resource utilization per each block is proportional to the logic occupation or required interconnection, the result-

ing partition will suffer from unbalanced resource utilizations or the critical resource over-utilizations.

In Chapter 4, we propose a new multilevel K-way partitioning approach which is aware of the resource utilization distribution. A new quality, *crowdedness*, is defined as a virtual complexity metric where the physical size and the local connectivity of a partitioned block are simultaneously considered in the form of a weighted sum. The partitioning solutions obtained by overall interconnection minimization have wide variances of local external degrees of the blocks, and this unbalanced crowdedness may yield unbalanced resource utilization and/or congestions in the following design steps. By minimizing the maximum crowdedness over the partitioned blocks, the proposed algorithms produce partitioning solutions where smaller blocks have denser external degrees and larger blocks have sparser external degrees. This can be viewed as an intelligent way of block balance relaxation based on connectivity. These results are not achievable with any previous conventional partitioning methods, even with minimization of maximum external degree.

- **Limited flexibility of multilevel partitioning paradigm.** Though the multilevel partitioning paradigm is relatively robust, it has limited flexibility; while the partitioning solution is refined over multiple levels, the problem instance at each level totally relies on the construction of the multilevel clustering tree. In particular, no knowledge that could have been obtained from the multilevel coarsening phase is used in the uncoarsening and refinement phase. Also, further improvements based on the trade-off between run time and quality have barely been introduced. Through the extensive experiments,

it is noticed that the increase of the number of levels does not provide reasonable trade-offs. The limitation of the flexibility of the multilevel partitioning is an obstacle for fundamental improvement.

In Chapter 5, we present a new improved version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree constructed during the coarsening phase; the multilevel uncoarsening phase involves *multirate unclustering*, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels.

Chapter 6 briefly summarizes this dissertation.

Chapter 2

Design Hierarchy Guided Multilevel Circuit Partitioning

In this chapter, we present a new multilevel circuit partitioning algorithm (`dhml`) which is guided by design hierarchy. In addition to flat netlist hypergraph, we use user design hierarchy as a hint for partitioning. This design hierarchy already has some implications on connectivity between logical blocks in the design. Using design hierarchy in partitioning is nontrivial since the hierarchical elements in design hierarchy do not necessarily have strong internal connectivity; hence we need to determine whether it is preferable to break up or preserve the hierarchical elements. In order to identify and select the hierarchical elements with strong connectivity, their Rent exponents are used. Then, the selected hierarchical elements are served as effective clustering scopes during the multilevel coarsening phase. The scopes are dynamically updated (enlarged) while building up a clustering tree so that the clustering tree resembles the densely connected portions of the design hierarchy.

We tested our algorithm on a set of large industrial designs in which the largest one has 1.8 million cells, 2.8 million nets, and 11 levels of hierarchy. By exploiting design hierarchy, our algorithm produces higher quality partitioning results than the state-of-the-art multilevel partitioner `hMetis`[39]. Furthermore, experimental results show that `dhml` yields significantly more stable solutions, which is helpful in practice to reduce the number of runs to obtain the best result.

2.1 Introduction

Circuit partitioning is a critical optimization problem in many areas of VLSI design automation because the partitioning solutions have a great impact on automatic placement and routing procedures. The attempts to solve this NP-complete problem have concentrated on finding heuristic algorithms which yield near-optimal solution in polynomial time. Some of the best known approaches include the iterative improvement methods such as Kernighan-Lin (KL), Fiduccia-Mattheyses (FM) algorithms and their variations[33, 43, 57, 4]. Recently, a new multilevel partitioning scheme has been introduced in order to improve partitioning results of iterative improvement approaches especially for bigger designs[2, 39, 41]. The multilevel partitioning has been shown to be the most effective approach to producing excellent partitioning quality in a comparatively short run time.

Generally a multilevel partitioning consists of 1) multilevel clustering (coarsening), 2) initial partitioning at the coarsest level, and 3) multilevel FM refinement with unclustering (uncoarsening). During the coarsening phase, the problem size is gradually reduced over the levels while capturing strong connectivity in the circuit netlist. Then, the initial partition at the coarsest level is propagated to lower lev-

els, at which FM partitioning is performed to improve the current initial partition inherited from the upper level. At each level, only a small number of passes are needed for FM refinement since the initial partition from upper level already has quite good quality.

In multilevel partitioning, the levels are determined in the coarsening phase while identifying and grouping the strongly connected vertices. Through the successive level-by-level clustering, a multilevel clustering tree $\mathcal{C}$ is constructed. The clustering tree $\mathcal{C}$ and design hierarchy tree $\mathcal{D}$ is similar in that both are the representations of multilevel hierarchical groupings. The proposed work is motivated by the observation that the well-grouped hierarchical elements in $\mathcal{D}$ can be used to guide the clustering tree construction. Since the design hierarchy already has some implications on connectivity between the logical blocks in the design in most cases, it can be beneficial to construct the clustering tree to be as similar to the design hierarchy as possible. However, we do not blindly follow every grouping in $\mathcal{D}$, but instead we identify and select some good hierarchical elements (i.e., the hierarchical elements with higher internal connectivity) to use them as clustering scopes. Rent exponents are used as quality indicators to find the good hierarchical elements, which are called positive scopes. After completion of each one-level clustering, a clustering scope is updated to a larger scope if clustering process in the scope turns out to be *saturated* so that the vertices in the current scope now have chances to be merged with others in the larger scope at the next level clustering. By this scope restriction, we expect entire clustering phase to produce a clustering tree which is biased to the well-grouped logical blocks in the design hierarchy.

For FPGA applications, a few partitioning methods utilizing design hierarchy

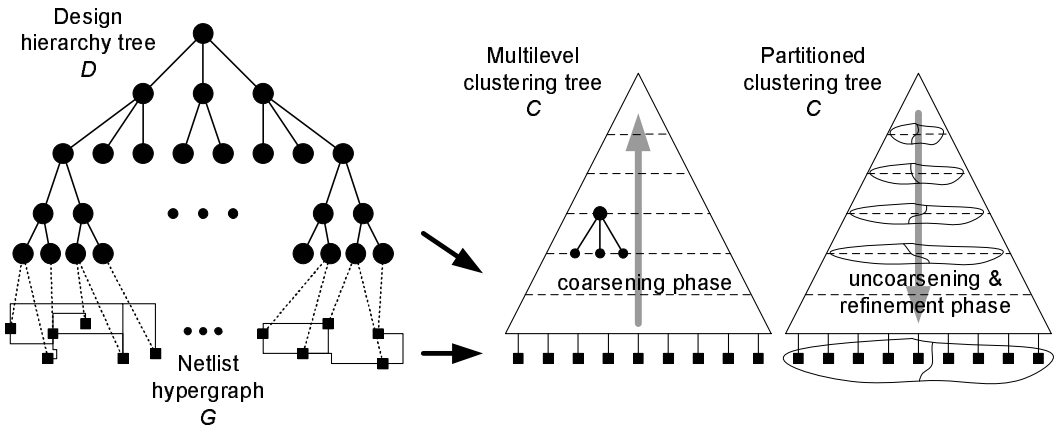


Figure 2.1: Design guided multilevel circuit partitioning problem.

have been reported recently[47, 7, 32]. They mainly focus on problem size reduction using design-based clustering. The hierarchical elements are selectively preserved if they are feasible — both size and pin count are smaller than the limit of each FPGA device. For non-feasible hierarchical elements, some operations are applied to intelligently break up the elements. Under the size and pin count constraints, usually their goal is to find a set of the good feasible blocks which maximizes device utilization, i.e., minimizes the number of FPGA's used. However, their frameworks are not directly applicable to general partitioning problems, and they may preserve the hierarchical elements with loosely connected internal cells unless the external pin count exceeds the upper bound. Moreover, they are not easily transformed into the multilevel scheme.

In this chapter, we proposes a new multilevel circuit partitioning, `dhml`, that benefits from design hierarchy. It takes a user design hierarchy as well as a netlist hypergraph, and construct a clustering tree that resembles the design hierarchy (See Figure 2.1). With this guidance from design hierarchy in multilevel clustering

phase, experimental results show that `dhml` yields higher quality solutions than a conventional multilevel partitioner, `hMetis`. Speed-up has been also achieved in a sense that near-optimal solutions are more frequently obtained in multiple runs since `dhml`'s partitioning solutions are more stable. An aggressively reduced number of levels in clustering phase also contributes to the speedup.

2.2 Problem Formulation

Definition 2.1. A circuit is modeled by a network of leaf cells represented by *hypergraph* $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of *leaf cells* and $\mathcal{E}$ is a set of *hyperedges (nets)*. A set of leaf cells which is a subset of $\mathcal{V}$ is defined as a *cluster*.

Definition 2.2. A design hierarchy provided by designer is represented by a rooted tree. A *design hierarchy tree* $\mathcal{D}$ is a collection of nodes and arcs such that a node is either a leaf cell or a hierarchical element which contains other nodes as children, and an arc represents the parent-child relationship between nodes. For a hierarchical element H, a corresponding cluster C_H is derived from H by simply taking the set of the bottommost leaf cells in H. We call the cluster C_H a *hierarchical cluster* derived from H.

Notation 2.1. For a given netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$ and design hierarchy $\mathcal{D}$,

1. $S(v) \equiv$ the physical size occupied by a leaf cell v.
2. $S(C) = \sum_{v \in C} S(v) \equiv$ the sum of sizes of leaf cells contained in a cluster C.
3. $|C| \equiv$ the number of leaf cells in a cluster C. For a hierarchical element H, $|H| \equiv |C_H|$.

4. $E(v) \equiv$ the number of nets incident on a leaf cell v.

5. $E(C) \equiv$ the number of external nets incident on a cluster C. This is often referred to as external degree or external pin count of the cluster C. For a hierarchical element H, $E(H) \equiv E(C_H)$.

Problem 2.1. Given a design hierarchy $\mathcal{D}$ and $G = (\mathcal{V}, \mathcal{E})$, the partitioning problem is to partition $\mathcal{V}$ into k disjoint subsets $V_1, \cdots, V_k$, with the objective of minimizing the cut-set size, i.e., the number of hyperedges that span multiple subsets. If $k = 2$, we call the problem bipartitioning.

2.3 Design Hierarchy

Figure 2.1 shows an example of design hierarchy. We use the Rent exponent as a quality indicator to determine which hierarchical element has a strong internal connectivity.

2.3.1 Rent's Rule and Rent Exponent

Rent's rule is an empirical formula which describes the general relationship between the number of cells and the number of external nets in a subcircuit (cluster). For a hierarchical element H,

$$E(H) = \bar{P}_H \cdot |H|^{r_H} \tag{2.1}$$

where r_H is Rent exponent or Rent parameter ($r_H \leq 1$) of H, $\bar{P}_H$ is the average number of pins per cell ($1/|H| \sum_{v \in C_H} E(v)$).

Rent's rule has been widely used for interconnection complexity estimation. Hagen et al.[36] defined the *intrinsic* Rent exponent to characterize the quality of

partitioning algorithms. The intrinsic Rent exponent of a given partitioning tree is the representative value of the quality measure for the corresponding partitioning algorithm. Also, there have been a few clustering algorithms to identify strongly connected cells using Rent exponent as a projected quality measure[54, 52]. In the Rent's rule based clustering algorithm, the locality of Rent exponent is more emphasized to select the best merging combinations from candidate neighbors.

Our approach is inspired by the combination of global and local connectivity information. Given a design hierarchy tree, we can estimate the global quality of the tree by computing a representative value of Rent exponent, $\bar{r}$. Contrary to the partitioning trees, design hierarchy trees usually do not have regular patterns in sizes and the number of nodes. Rent exponent extraction method in [36] may not be feasible because data points gathering for linear regression is not totally controllable. Hence we have used the average value of all Rent exponent of the hierarchical elements in $\mathcal{D}$ weighted by cluster sizes, i.e., $\bar{r} = (1/\sum |H|) \cdot (\sum r_H |H|)$. The representative value obtained from the above is not so useful unless it is combined with local measure. As used in [54, 52], local Rent exponent is beneficial to exploit local connectivity information.

Let $P(H)$ be the total number of pins of cells in H, i.e., $P(H) = \sum_{v \in C_H} E(v)$. Also, let $I(H)$ be the total number of internal pins in H, i.e., $I(H) = P(H) - E(H)$. Since $\bar{P}_H = (I(H) + E(H))/|H|$, from equation (2.1),

$$r_H = \frac{\ln E(H) - \ln((I(H) + E(H))/|H|)}{\ln |H|} \tag{2.2}$$

$$= \frac{\ln(E(H)/(I(H) + E(H)))}{\ln |H|} + 1 \tag{2.3}$$

From the equation (2.3), we note that Rent exponent also captures some information on the internal connectivity. It is obvious that if all pins contribute

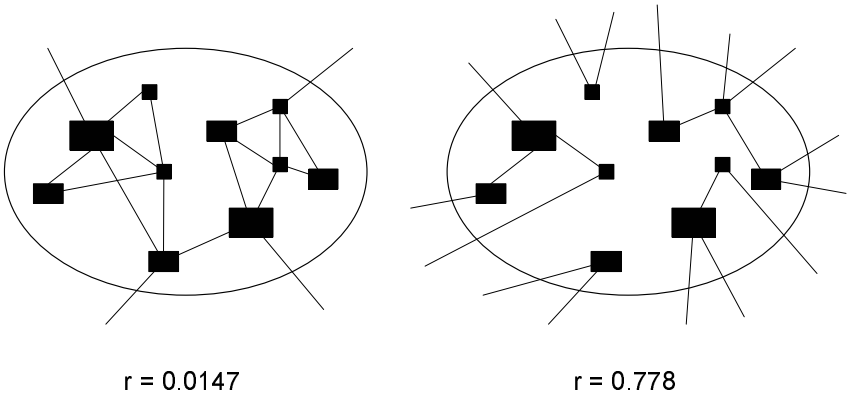


Figure 2.2: Implication of local Rent exponent.

to the external nets, i.e., $E(H) = P(H)$, $r_H = 1$ which is the maximum value. From the viewpoint of internal connectivity, small Rent exponent implies relatively high connectivity inside and large Rent exponent implies low connectivity. In [47], a similar measure — S/T quality (ratio of size to external pin count) — was used to estimate the connectivity quality of hierarchical elements. However, it does not capture the internal connectivity, which is more helpful to identify the hierarchical elements that have more strongly connected cells inside.

As shown in Figure 2.2, a hierarchical element H with small r implies that it contains relatively more strongly connected cells inside. Thus it is preferable to preserve the internal connectivity. On the other hand, a hierarchical element H with large r implies that it has relatively more connections with outside cells, which means it is preferable to remove the grouping by H so that the cells in H can have chances to be chosen as strongly connected neighbors from outside of H.

For a hierarchical element H, the weighted average of Rent exponents, $\bar{r}$ is used as a threshold value to determine whether the corresponding Rent exponent r_H is small or large. A hierarchical element H is said to be a *positive* scope if

$r_H < \bar{r}$, a *negative* scope otherwise. With the guidance of preliminary knowledge of connectivity information from design hierarchy, multilevel clustering is performed while restricting the clustering scopes to good hierarchical groupings — positive scopes.

2.4 Multilevel Partitioning

In this section, we provide a multilevel clustering algorithm which is guided by the Rent exponents which imply the local connectivity quality of the design hierarchy. Then, the entire partitioning algorithm, `dhml`, is presented.

2.4.1 Design Hierarchy Guided Clustering

As shown in Figure 2.1, coarsening phase of the multilevel partitioning consists of successive bottom-up clustering procedures from a set of leaf cells. During the coarsening phase, large nets are contracted to smaller nets and a sequence of successively smaller hypergraphs are constructed. Thus, several vertices at the current level are merged together into a bigger vertex at the upper level, eventually forming a k-level tree $\mathcal{C}$.

The main purpose of multilevel clustering is to create a small hypergraph such that a good bisection of the small hypergraph is not significantly worse than the bisection directly obtained from the original hypergraph[39]. This quality preservation ensures that the following top-down refinement does not need much effort to improve the initial partition inherited from the upper levels. Although best initial partition at the coarsest level does not always guarantee the best partition at the finest level with leaf cells, there are more chances to reach a higher quality final

Procedure `construct_cluster_tree`

Input: bottommost netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$,
h-level design hierarchy tree $\mathcal{D}$
Output: k-level clustering tree $\mathcal{C}$

1. Extract scope tree $\mathcal{D}'$ from $\mathcal{D}$
2. **for** each leaf cell $v \in \mathcal{V}$ **do**
3. Determine a clustering scope $H(v)$ in $\mathcal{D}'$
4. $G_0(\mathcal{V}_0, \mathcal{E}_0) = G(\mathcal{V}, \mathcal{E}),\ \ k = 0$
5. **do**
6. $G_{k+1} =$ `cluster_one_level`(G_k)
7. $k = k + 1$
8. **while** $|\mathcal{V}_k| > \alpha$ and $|\mathcal{V}_k|/|\mathcal{V}_{k-1}| < \beta$

Figure 2.3: Clustering tree construction procedure.

solution if we construct a higher quality clustering tree. Fortunately we have an additional grouping information in user design hierarchy tree $\mathcal{D}$, which is originally based on functional decomposition. Design hierarchy $\mathcal{D}$ and Rent exponents of hierarchical elements in $\mathcal{D}$ give a guidance for the multilevel clustering procedure.

Clustering is defined as the merging process of the existing vertices to form a smaller number of bigger vertices. In multilevel partitioning schemes published, pairwise merging has been widely used[39, 41, 2]. We have performed extensive experiments with various clustering methods, and FC(First Choice) coarsening proposed in [41] turned out to be the most effective. Thus, we are using a connectivity cost and merging policy similar to FC in `hMetis`. Even though we are using the same idea as the one in `hMetis` to form bigger vertices, entire clustering scheduling is quite different because of the existence of guidance from design hierarchy $\mathcal{D}$.

Figure 2.3 summarizes our multilevel clustering tree construction procedure. As an initialization process, the original design hierarchy tree $\mathcal{D}$ is restructured such

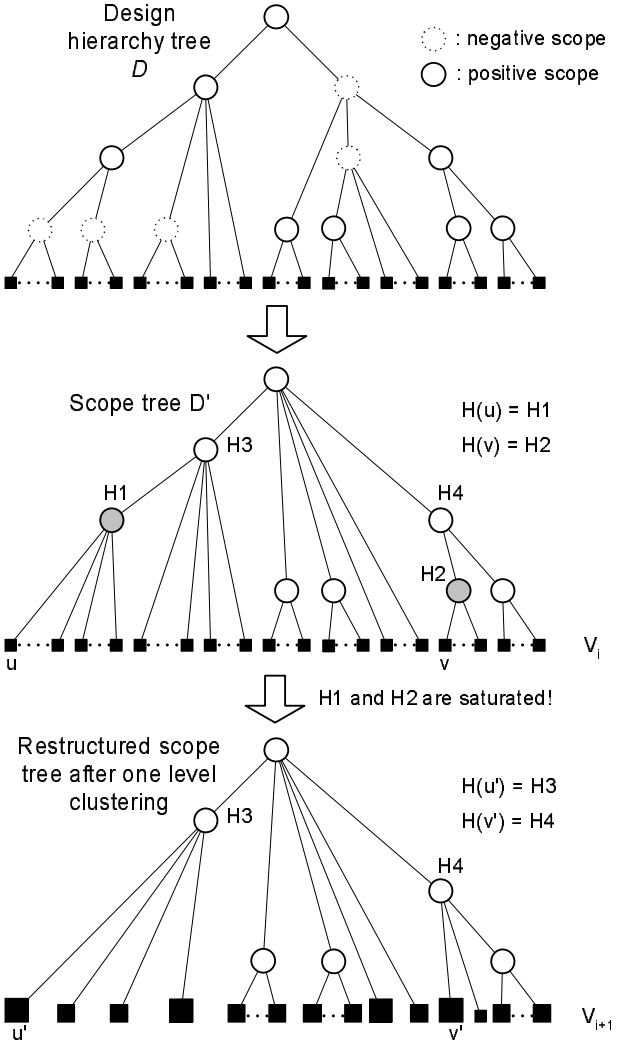


Figure 2.4: Design hierarchy restructuring.

that all the negative scopes are removed from $\mathcal{D}$ (See Figure 2.4). The resulting tree only with the hierarchical elements which are positive scopes is called *scope tree*. In general, the vertices in a negative scope have more freedom to be merged with others outside of the scope. Meanwhile, the vertices in the positive scope are restricted to be merged with those within the scope. Next, for each leaf cell v, we determine a *clustering scope* in $\mathcal{D}'$ (Step 2, 3). A clustering scope of v is the closest (and smallest) ancestor of v in D whose Rent exponent is smaller than $\bar{r}$, and denoted by $H(v)$. Hence, initially the parent of v in the scope tree $\mathcal{D}'$ is chosen as $H(v)$. If there is no positive scope for v in $\mathcal{D}$, the root of $\mathcal{D}$ is assigned to $H(v)$; no clustering scope restriction is applied for such v.

Then, starting from the bottommost netlist hypergraph, we successively construct the upper level hypergraphs until the number of vertices is small enough or clustering process is *saturated*, i.e., no more significant reduction is available (Step 4–8). In step 8, α and β represent a desirable size at the coarsest level and a threshold value of slow reduction rate respectively. In our experiment, $\alpha = 100$ and $\beta = 0.9$ have been used.

The main clustering procedure is presented in Figure 2.5. First, we arrange the vertices in $\mathcal{V}_i$ so that cluster v cannot be matched before u if $H(v)$ is a proper ancestor of $H(u)$ (i.e., the set of leaf cells in $H(v)$ contains all the cells in $H(u)$ and $H(v)$ is a larger scope than $H(u)$) The vertices whose clustering scopes with level (height) l in $\mathcal{D}$ are grouped together in $\mathcal{W}_l$ (Step 1).

Next, the matchings are performed from the lowest level $\mathcal{W}_1$ at which the scopes are the smallest. For each level of the scope, we randomly select an anchor cluster v which has not yet been matched to form one of the upper level vertices.

Procedure `cluster_one_level`

Input: hypergraph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$
Output: contracted hypergraph G_{i+1}

1. Compute $\{\mathcal{W}_l \mid \mathcal{W}_l = \{v \mid v \in \mathcal{V}_i, H(v)\text{'s level is } l\}\}$
2. **for** $l = 1$ to h **do**
3. **while** there exist unmatched vertices in $\mathcal{W}_l$ **do**
4. Randomly select an unmatched v in $\mathcal{W}_l$
5. Find $w \in \mathcal{V}_i$ that maximizes `clus_cost`(v, w) such that $H(w) \subseteq H(v)$
6. **if** there is no such w
7. Create a singleton cluster v' in $\mathcal{V}_{i+1}$
8. **else if** w is unmatched
9. Create a cluster v' in $\mathcal{V}_{i+1}$ with (v, w)
10. **else if** w is matched in existing $v' \in \mathcal{V}_{i+1}$
11. Append v to v'
12. $H(v') = H(v)$
13. Remove saturated hierarchical elements in $\mathcal{D}'$ and update $H(v')$ accordingly

Figure 2.5: Main clustering procedure.

For the cluster v, we consider the neighbors which are connected to v by some nets in $\mathcal{V}_i$ and are located inside of $H(v)$. We have an upper bound size μ for the resulting upper level vertex. In step 5, with the restricted scope and the size upper bound, the best target vertex w is chosen that maximizes a `clus_cost`(v, w) $= \sum_{e \in \mathcal{E}_i, e \in \{e \mid v \in e, w \in e\}} 1/(|e| - 1)$ which is adopted from [39]. Using the restricted scope, only the potentially good neighbors in $H(v)$ are searched to be merged with v. If the best neighbor w has not yet contributed to form any upper level vertices, v is merged with w and a new vertex $v' \in \mathcal{V}_{i+1}$ is created (Step 8–9). If the best neighbor w has been already matched with others and contributed to form an upper level vertex v', v is appended to v' which w belongs to (Step 10–11). In case that there is no such best neighbor for v, a new singleton vertex v' only with v is created

(Step 6–7).

Like FC coarsening in [39], only-unmatched-neighbor condition is relaxed, and even the matched neighbors are also considered unless the size of the resulting upper vertex violates the maximum allowable size μ. In our experiments, μ has been set to $b \times S_{total}$, where b is the balance ratio parameter for a partitioning problem and $S_{total} = \sum_{v \in \mathcal{V}} S(v)$. If both matched and unmatched vertices are considered as neighbors, the number of vertices in the successive coarse hypergraphs may decrease by a large factor, potentially limiting the effect of multilevel refinement. Hence, usually the reduction rate is controlled by a certain constant in the conventional multilevel partitionings, e.g. 1.7 to ensure sufficiently many levels in a clustering tree for refinement procedure to effectively improve the quality. However, we have removed the reduction rate control to take full advantage of the guidance of design hierarchy, and the matching (merging) procedure is performed until every lower level vertex is contributed to an upper level cluster. As a result, the number of levels is usually significantly less than that of the conventional multilevel partitioning. Our experimental results show that this does not degrade the quality since our clustering procedure builds up a clustering tree with better connectivity quality in spite of having fewer levels due to the correct guidance from design hierarchy.

The clustering scope of a newly created vertex v', $H(v')$, must be updated for the next round clustering on $\mathcal{V}_{i+1}$. It is obvious that $H(v')$ is set to $H(v)$ for a new vertex v' created in step 7 and 9 since $H(w) \subseteq H(v)$. Even in the case that v is appended to an existing upper level vertex v' (Step 11), $H(v') \subseteq H(v)$ is guaranteed due to the initial ordering by scope level in step 1, hence we choose $H(v)$ for the new clustering scope of v' (Step 12).

After the completion of one-level clustering, the existing clustering scopes are examined and updated for the next level clustering in step 13. As the global saturation condition is examined in `construct_cluster_tree`, for each clustering scope, local saturation condition is checked and the scope is removed from the scope tree to enlarge the scope if necessary as shown in Figure 2.4. For a clustering scope X in $\mathcal{D}'$, let $C_{X,i}$ be $\{v \in \mathcal{V}_{i+1} | H(v) = X$ after i-th level clustering$\}$. Then, a clustering scope X is said to be *saturated* at level i and removed if either $|C_{X,i}| \leq \alpha(X)$ or $|C_{X,i}|/|C_{X,i-1}| \geq \beta(X)$ (in our experiments, $\alpha(X) = \lceil S(X)/\mu \rceil$ and $\beta(X) = 0.7$). If X turns out to be saturated after the one-level clustering, for every cluster v in $C_{X,i}$, $H(v)$ is updated with the parent of X in the scope tree $\mathcal{D}'$, which is larger but still is a good grouping in $\mathcal{D}$. Whenever one level clustering is done, we check and remove the local saturation in this manner until the global saturation condition holds.

2.4.2 Design Hierarchy Guided Multilevel Partitioning

Figure 2.6 describes `dhml`, our multilevel bipartitioning algorithm using design hierarchy guided clustering. As a first step, the algorithm computes the Rent exponent for each hierarchical element. Then, as shown in the previous section, the design hierarchy tree $\mathcal{D}$ is used to guide multilevel clustering. After the completion of the multilevel clustering, we have a k-level clustering tree. At the coarsest level k, FM bipartition is applied for max times with different random initial bipartitions, and the best cut quality partition is chosen to be propagated down to the lower levels as an initial partition (In our experiments, $max = 20$). The initial partitioning phase (step 3–5) is very fast since there are only a few vertices to be partitioned at the

Algorithm `dhml`

Input: netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$,
design hierarchy tree $\mathcal{D}$
Output: bipartition $P_0 = \{V_1, V_2\}$

1. Perform Rent exponent computation on $\mathcal{D}$
2. `construct_cluster_tree`(G, $\mathcal{D}$)
3. **for** $i = 1$ to max **do**
4. Generate a random initial bipartition R on G_k
5. Q_i = `FMbipartition`(G_k, R)
6. P_k = best bipartition among Q_i's
7. **for** $i = k - 1$ down to 0 **do**
8. P_i = `FMbipartition`(G_{i+1}, P_{i+1})

Figure 2.6: The `dhml` multilevel partitioning.

coarsest level. In step 7 and 8, uncoarsening and FM refinement is performed using the current best partition from the upper levels as an initial partition. The number of passes to improve the current partition is only one or two in most cases because the initial partition from the coarser levels has already good quality.

We also can extend the algorithm `dhml` to multi-way partitioning by applying this bipartitioning recursively. For each bipartitioning, a partial design hierarchy tree is extracted from the original entire design hierarchy tree $\mathcal{D}$, in which the leaf cells are the cells that belong to the current partition. This partial tree is used to guide the sub-partitioning in the same way described previously.

2.5 Experimental Results

We implemented our algorithm in C++/STL and evaluated the performance on six large scale industrial circuits, which are real circuits used in industry.[1] The

[1]Note that we cannot use the standard partitioning benchmark circuits from MCNC and ISPD-98 since the design hierarchy information is not given. Moreover, the industrial circuits we use are

characteristics of the design hierarchies and the netlist hypergraphs for these circuits are shown in Table 2.1, where the largest circuit `ind6` has about 1.8 million cells, 2.8 million nets, and 11 levels of hierarchy. The fourth column shows the height of each design hierarchy tree and the number of hierarchical elements. We first describe the stability of our algorithm, and then the cut quality is shown as the number of partitions varies. Finally, the preservation of design hierarchy is discussed.

Table 2.1: The characteristics of the circuits.

Circuit	No. cells	No. nets	h/No. hier. nodes
ind1	15186	19152	6/302
ind2	136340	183340	9/10427
ind3	224908	187595	5/57590
ind4	414633	414013	13/94796
ind5	1213105	1317889	13/33277
ind6	1841147	2788461	11/35449

As pointed out in [4], a common weakness of the partitioning methods based on iterative improvement is that the solution quality is *not stable*. This instability is inherent to conventional multilevel partitioners as well since they also use move-based approaches which depend on the initial solutions. However our algorithm which is guided by design hierarchy shows more stable solution ranges while having better minimum solution as shown in Figure 2.7. Our experiments indicate `dhml` will have more chances to achieve near-optimal solutions in a smaller number of runs. Also, the average solution is very close to the minimum solution, which is useful in real CAD tools because hundreds of runs cannot be performed.

Table 2.2 summarizes the cut set size comparison of `dhml` and `hMetis`.[2]

significantly larger than the standard benchmark circuits.

[2]Note that 256-way partitioning is meaningful since systems with more than 200 FPGA's are commercially available these days.

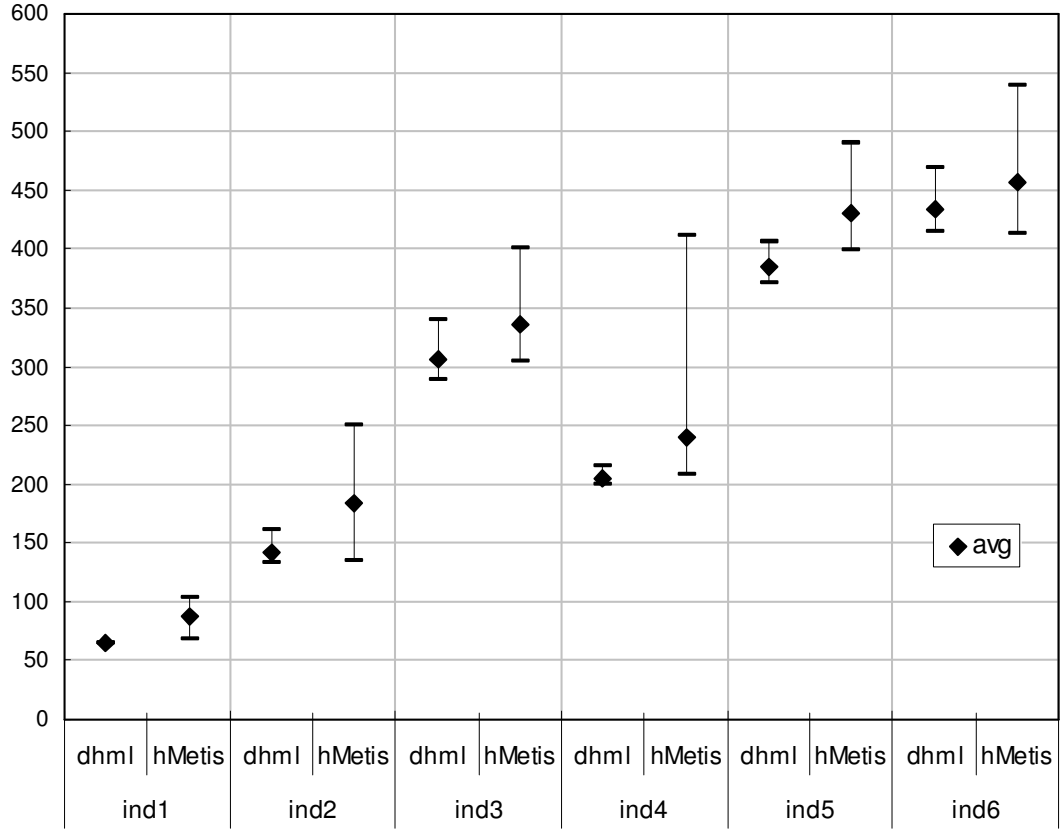


Figure 2.7: The ranges of partitioning solutions from 10 runs of each partitioner (Cut set sizes of 2-way partitionings for ind1, ind2, ind3, and ind4, and cut set sizes of 16-way partitionings scaled by 1/20 for ind5 and ind6 are shown).

Table 2.2: Minimum cut set size comparison of `dhml` *vs.* `hMetis` with 5% balance ratio at each bipartitioning.

Circuit	2-way		16-way		256-way	
	`dhml` (5 runs)	`hMetis` (10 runs)	`dhml` (5 runs)	`hMetis` (10 runs)	`dhml` (5 runs)	`hMetis` (10 runs)
ind1	64	69	437	483	–	–
ind2	133	134	1203	1294	14633	16137
ind3	292	305	1454	1551	7450	7508
ind4	202	208	3394	3498	12013	13999
ind5	1376	1352	7410	7950	22474	24454
ind6	55	56	8275	8265	33472	35075

For fair comparison, `hMetis` results are based on FC coarsening and FM refinement options.[3] As shown in the table, `dhml` produces up to 16% improved results in terms of the minimum cut set sizes in half the runs of `hMetis`. The quality improvements are more prominent when the number of partitions is relatively large. For large scale designs, a small number of partitions blunt the favorable effect from the guidance of design hierarchy because the resulting partitions are too coarse to benefit from the clustering guided by the hierarchical elements whose sizes are much smaller than the partition sizes. Also, note that the balance constraint is actually more relaxed as the number of partitions increases since the imbalance at the top level bipartitioning can be accumulated to the lower level sub-bipartitionings. This loose balance constraint for a large number of partitions enables `dhml` to produce relatively higher quality solutions than it is applied to only a small number of partitions since it helps the well-grouped clusters (possibly quite irregular in sizes) avoid being forced to split

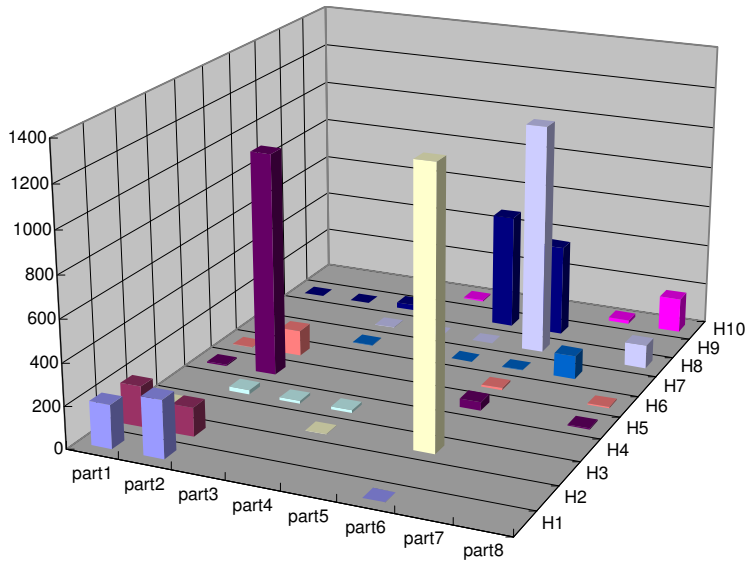
[3]The simplified version of hMetis, shmetis, uses default options of HFC (hybrid FC which is a variation of FC), FM refinement, and V-cycle. However, shmetis do not show any significantly different results from hMetis with the options we used. The post-processing called V-cycle in shmetis also can be applied to dhml to enhance the final result, but the impact on the quality was very little for both hMetis and dhml.

up for balancing.

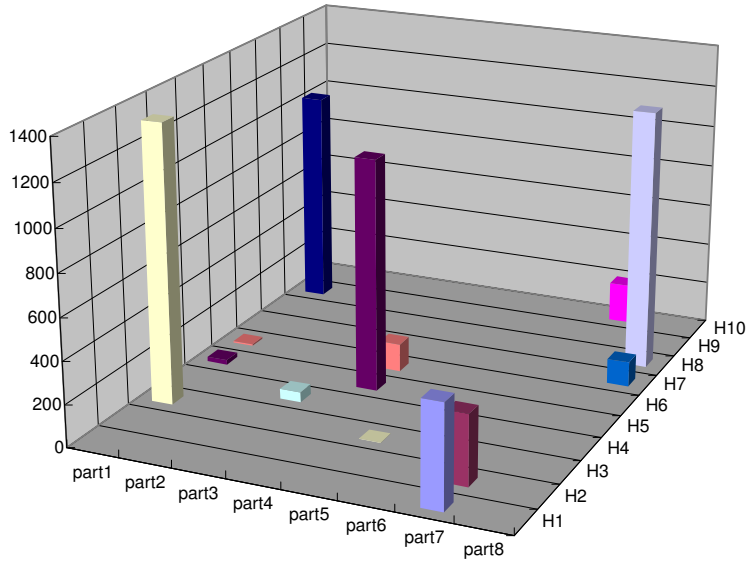
As many researchers noticed, the direct k-way partitioning is very susceptible of being trapped into a local minima and requires prohibitively large amount of memory even though it is the fastest way of partitioning and beneficial to better balancing. Our experiments also agreed with this observation and the direct k-way version of `hMetis` (khmetis) produced the solutions averagely 56% worse than the recursive version in the same number of runs; hence the results of khmetis have been dropped from the comparison.

In the case of `ind1`, the minimum cut set size of 64 was obtained in every run of `dhml`. In addition, due to the design hierarchy guided clustering, all the minimum cuts have been achieved at the coarsest level (i.e., no further improvement was needed in the refinement phase), where the problem size has been reduced to 1/150. In most cases, the initial partitions at the coarsest level show 20%–50% smaller cut sets than `hMetis`, which justifies the superior quality of the multilevel clustering tree guided by design hierarchy. This implies that the proposed design hierarchy guided multilevel clustering may be used as a stand-alone preprocessing step to reduce the problem size for partitioning of very large scale circuits without significant sacrifice of quality. Also, the number of levels in `dhml` is reduced to 55%–75% of that in `hMetis` while the number of passes for FM refinement at each level is not increased.

Table 2.3 reports the CPU times for 10 runs of 2-way partitioning. The third column represents the CPU times for `dhml` without design hierarchy guidance option, which has the same functionality and time complexity as `hMetis`. As shown in the table, the run time for `dhml` with design hierarchy guidance is smaller than that of



(a)



(b)

Figure 2.8: Preservation of design hierarchy for 8-way partitioning (ind1); Distribution of the leaf cells in each of the hierarchical elements after partitioned by (a) `hMetis` (cut set size = 288) and (b) `dhml` (cut set size = 228)

Table 2.3: CPU times for 2-way partitionings. Each value reports the total amount of time required by each of the partitioners for 10 runs. (the times are in seconds on an Ultra Sparc 5 @360MHz).

	dhml		hMetis
Circuit	dh-guided	no dh-guided	
ind1	12.3	16.7	10.2
ind2	133.2	177.1	115.7
ind3	232.0	255.5	213.1
ind4	477.2	511.6	439.7
ind5	1625.6	1752.4	1483.6
ind6	4377.4	4576.5	4120.3

`dhml` without design hierarchy guidance even though there are a few additional steps to use design hierarchy information. This indicates that clustering scope restriction and level reduction contribute to the runtime reduction. However, not because of the difference in algorithmic complexity, but because of the difference in implementation details, `hMetis` is still slightly faster than `dhml`.

Lastly, the graphs in Figure 2.8 visualize a typical comparison of the preservation of design hierarchy in terms of the leaf cell distribution after partitioning. In these graphs, 10 hierarchical elements whose leaf cells have been distributed over two or more parts in `hMetis` are randomly chosen (Figure 2.8 (a)), and their distribution in `dhml` is depicted again in Figure 2.8 (b). It is obvious from this example that `dhml` better preserves the groupings in design hierarchy while still yielding a higher quality solution (26% smaller cut set size in this example) than `dhml`. This design hierarchy preservation can be desirable for some other physical design issues (e.g. circuit performance, predictable place & route, etc.)

2.6 Conclusion

A new multilevel partitioning framework that takes advantage of user design hierarchy has been presented. As a guidance of design hierarchy, clustering scope restriction is used to construct a multilevel clustering tree. The clustering scopes are selectively determined by Rent exponent computation and updated dynamically while the clustering tree is being built up. Due to the benefit from the guidance by the design hierarchy which has implications on connectivity between functional blocks, our proposed algorithm generates better multilevel clustering tree while the number of levels is aggressively reduced. Our experiments on large scale real circuits show that `dhml` yields more stable and higher quality partitioning solutions in fewer runs than `hMetis`.

Chapter 3

Stable Multiway Circuit Partitioning for ECO

We propose a new *stable* multiway partitioning algorithm, where stability is defined as an additional quality of a partitioning solution. The stability of a partitioning algorithm is an important criterion for a partitioning based placement to achieve timing closure through the repetition of the placement procedure [63]. Given a previous partitioning result P^* on an original netlist hypergraph H^* and a partially modified netlist hypergraph H, a new cost function with similarity factor is defined to produce a new partition P on H which is *similar* to the original partition P^*. The proposed algorithm is the first approach that quantifies the degree of similarity of a current partition to the original partition using similarity cost. Our goal is to build a new partition in a relatively short run time, whose cut quality is not much degraded from that of the original partition P^* while it preserves as much of the previous groupings in P^* as possible. We extend the proposed algorithm to the

multilevel paradigm using restricted coarsening, and the multilevel implementation of the algorithm shows a dramatic speed-up without sacrificing the desired qualities of the partitioning solutions. The proposed partitioner is especially beneficial to engineering change order (ECO) applications, where partial modifications of a netlist are handled by the incremental methodology in a design iteration cycle. Our approach helps ECO placers maximize the incremental capability since the portions to be re-placed are minimized. Experimental results show that both the flat and multilevel version of the proposed algorithm achieve a high quality partition comparable to a state-of-the-art multilevel partitioner hMetis [41], while many portions of the groupings in the previous partition are preserved in the current partition. The trade-off between similarity and cut quality with respect to a varying similarity coefficient is also shown.

3.1 Introduction

As mentioned in Chapter 1, millions of the cells cannot be handled in a flat mode any more due to the limitation of computation power and memory space, and concurrent engineering of the individual subcircuits can shorten design turnaround time. As a result, a circuit needs to be partitioned into several blocks where the massive amount of data is broken into manageable sizes. Also partitioning techniques are often embedded in the large scale placement procedure to determine the optimal global positioning.

Multiway partitioning can be implemented in two different methods; one is applying a bipartitioning algorithm (e.g. FM algorithm) recursively (recursive bipartitioning, RBP) [43, 33], and the other is a direct K-way partitioning, an extension

of the FM algorithm to multiway partitioning [57, 55]. Though direct multiway partitioning tends to produce relatively worse quality solutions and requires more amount of memory than RBP technique, it shows faster runtime and higher balance management capability. Maintaining a global view over entire blocks renders the direct multiway partitioning suitable for quick partitioning refinement or incremental partitioning in engineering change order (ECO) situations, where partial modifications of a circuit are handled by the incremental methodology in the design iteration cycle.

The move-based partitioning techniques heavily rely on multiple runs in order to obtain a best solution among them. The different runs with different random initial partitions lead to a wide spectrum of solutions, from which the best quality solution is picked [4]. Even the multilevel partitioning paradigm [39, 41], which is known to be the most effective approach to producing excellent partitioning quality in a relatively short runtime, is subject to this *instability*, since FM is still used as an underlying partitioning algorithm at each level. The instability caused by randomness gives us opportunities to find better solutions through the multiple runs. However, as pointed out recently [68, 63], a lack of stability ironically yields a critical weakness in ECO situations for large scale designs including the reiteration of placement procedures for timing closure; very minor changes to a netlist can produce radically different results, hence it is difficult to achieve timing closure in the placement iterations. The instability of the min-cut based placement procedure is due to the limitation of the partitioning algorithms which is *unstable* in the multiple runs.

A small number of research results in the area of ECO (or incremental) algo-

rithms for physical design have been reported recently [25]. Even though there have been a few works on floorplanning [28], placement [23, 51], and FPGA placement and routing [61, 62], very little work has been done in incremental partitioning. In this chapter, we propose a novel incremental algorithm for multiway partitioning to support ECO applications, taking into account the *stability* of a partition.

Given a previous partitioning result P^* on an original netlist hypergraph H^* and a partially modified netlist hypergraph H, a new cost function with similarity factor is defined to produce a new partition P on H which is *similar* to the original partition P^*. The new similarity factor measures how much the current partition resembles the previous partition, i.e., how many of the cells that were grouped in one block remain grouped in a same block. Using a new cost function combined with similarity factor, stability is defined as a quality of a partition, and cut quality and similarity are considered simultaneously.

The proposed partitioning technique is especially beneficial to incremental(or ECO) placers within each resulting blocks, since it tries to minimize the portions to be re-placed implying their incremental capability to handle re-usable portions are maximized, while the inter-block connections are still taken care of. There are also several other applications that can benefit from the use of stability. Suppose a partition obtained from any kind of grouping criteria, which is not necessarily in a regular or balanced form, e.g., logical blocks in design hierarchy, clusters having preferable delay performance, partitioning results with a different objective or even with a different number of blocks, etc. Our approach provides a systematic methodology to make the final partition resemble the original grouping so that the desirable properties in the original groupings are preserved without detailed knowledge on the

criteria that the original grouping was obtained from.

A simple approach has been suggested as a preliminary solution for incremental partitioning in [25]. They demonstrated a trade-off between cut quality and runtime with respect to the threshold value which determines the range of the adjacent neighbors of the modified cells that are allowed to move. Higher threshold values allow more indirect neighbors of the modified cells to move, while zero threshold value prohibits any moves of other cells except the modified cells themselves. With zero threshold value, a resulting partition will be very similar to the original one since all the non-modified cells remain in their previous blocks. However, as conceived naturally, lower threshold values (especially with more modification rate, e.g., $\geq 10\%$) render the cut qualities of the solutions very far from near-best solutions even though runtime is much shorter thanks to the greatly reduced problem size. On the other hand, if a little higher threshold value is used (i.e., larger amount of the circuits are to be rearranged), most of the cells in the hypergraph are allowed to move and they form a totally different result from the original partition.

Another intuitive way of stable partitioning is to start the iteration of moves not at a random partition, but at the one derived from the previous partition, and follow the same procedure of the normal FM algorithm. In this case, there is no problem size reduction effect, but speed-up is still achieved in a sense that multiple runs to obtain a best partition are not necessary. However, the resulting partition is not guaranteed to be similar to the previous partition, since the sequence of the cell moves is driven solely by cut quality.

Our contribution can be summarized as follows; 1) Assuming the original partition is the best quality solution for the original circuit, for a partially modified

circuit, a near-best solution is quickly achieved starting from the original partition as an initial partition. Unlike the preliminary work above, a global view of the entire circuit is maintained over the entire blocks throughout the partitioning procedure. The need for multiple runs to obtain a best solution is removed and only a small number of passes is needed in the one run if we emphasize similarity. 2) We provide a novel scheme to quantify the similarity of the current partition to the original partition. As opposed to the conventional partitioners without any consideration of the similarity, our approach produces a stable solution that is quite similar to the original partition with a little sacrifice of cut quality (or possibly with a better cut quality). A stable partitioning solution obtained from an incremental partitioner has a great impact on the following ECO placement and routing steps. The larger amount the cells are aggregated as in the previous partition, the smaller portions of the blocks are to be rearranged by the incremental placers and routers. 3) Using the proposed algorithm as the underlying partitioner for each level of refinement, our approach is extended to the multilevel paradigm. Restricted coarsening used in V-cycle [39] is modified for the construction of a multilevel clustering tree. The multilevel implementation of the algorithm shows a dramatic speed-up without sacrificing the desired qualities of the partitioning solutions of the flat version.

3.2 Preliminaries

The notations used throughout this chapter are defined below. The basis of the notations comes from [55] and definitions to constitute our cost function are also described.

Definition 3.1. A circuit is modeled by a netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$, where $\mathcal{C}$

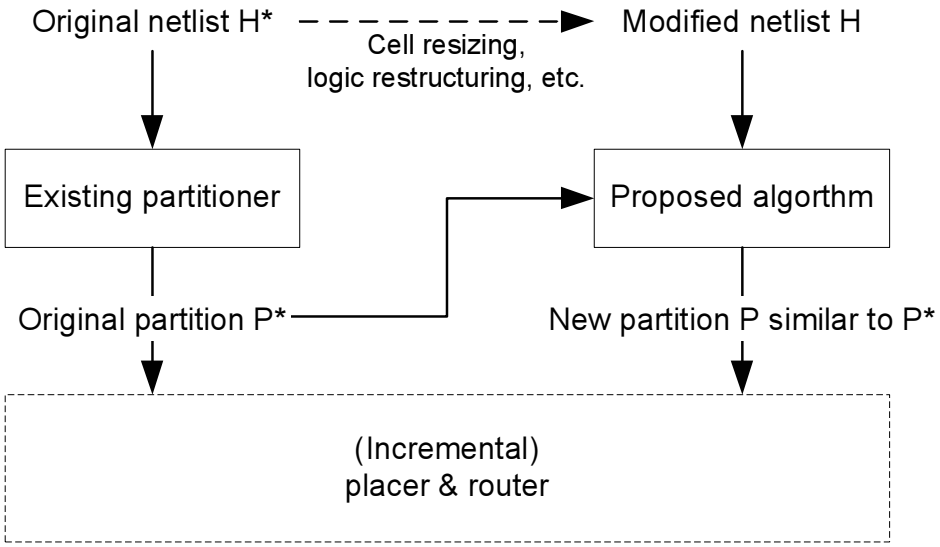


Figure 3.1: Flow chart for the proposed algorithm.

is a set of *cells* with associated sizes and $\mathcal{N}$ is a set of *nets* (hyperedges) connecting two or more cells. A *pin* is a connection point between a cell and a net.

Definition 3.2. A *K-way partition* of a hypergraph is described by the K-tuple, $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$ where $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for $i \neq j$ and $\cup_i \mathcal{A}_i = \mathcal{C}$. $\mathcal{A}_i$ is said to be the i-th block of the partition.

Definition 3.3. Given a circuit hypergraph $H = (\mathcal{C}, \mathcal{N})$, the K-way partitioning problem consists of finding a partition of K blocks ($K \geq 2$) such that the interconnections between the blocks are minimized while the block sizes are constrained to a certain size.

The objective cost function to quantify the inter-block interconnection varies; The most popular cost function is the cutset size, the cardinality of the set of the nets which are *cut*, i.e., the number of nets having at least one cell in more than one block. In our work, a cut net is assigned a value $i - 1$ if the number of blocks the

net straddles is i, and the sum of these values of cut nets constitute the cut cost. Similarly, the sum of external degree (SOED) of each block can be used as a cut cost. In this case, a cut net is assigned a value i if the number of blocks the net straddles is i.

Notation 3.1. For a given hypergraph $H = (\mathcal{C}, \mathcal{N})$ and a given block $\mathcal{A}_i$,

(1) $S(C) \equiv$ the size of a cell C.

(2) $|\mathcal{A}_i| \equiv$ the number of cells in $\mathcal{A}_i$.

(3) $n_C \equiv$ the number of nets incident on a cell C (the number of pins on C).

(4) $p = \max_{C \in \mathcal{C}}(n_C)$, $p_{avg} = \sum_{C \in \mathcal{C}}(n_C)/|\mathcal{C}|$

In this chapter, *the original partition*, denoted by $P^* = (\mathcal{A}_1^*, \mathcal{A}_2^*, \cdots, \mathcal{A}_{K^*}^*)$, is the resulting partition from the original hypergraph H^* using one of any existing partitioning algorithms. The original partition is also called the *golden partition* in this chapter since it is assumed to be the most desirable solution for the original hypergraph H^*. Figure 3.1 shows the basic flow in which our algorithm is applied using P^*.

For a series of partitions, we define *stability* as follows.

Definition 3.4. *Stability* is defined as the quality that the ensuing partition P on the modified hypergraph H is similar to the original partition P^* on the original hypergraph H^*. The terms stability and similarity are used interchangeably in this chapter. For a partition to be more stable with respect to P^* (or more similar to P^*), the cells in the current partition need to have as many of *block neighbors* as possible.

Definition 3.5. *Block neighbors* of cell C, $BN(C)$ is defined as the set of the cells that *were* placed in the same block following an original partition P^*. For a cell

$C \in \mathcal{A}_i^*$,

$$BN(C) = \mathcal{A}_i^* \cap \mathcal{C}$$

Note that C is a block neighbor of itself. Also, block neighbors of cell C in $\mathcal{A}_j$ are defined as

$$BN_{\mathcal{A}_j}(C) = BN(C) \cap \mathcal{A}_j = \mathcal{A}_i^* \cap \mathcal{A}_j$$

Missing block neighbors of cell C in $\mathcal{A}_j$ are defined as the set of cells that were placed in the same block in P^*, but now are in a block other than $\mathcal{A}_j$ in P

$$\overline{BN_{\mathcal{A}_j}}(C) = BN(C) - BN_{\mathcal{A}_j}(C) = \sum_{k \neq j} BN_{\mathcal{A}_k}(C)$$

It is convenient to introduce a notation B_j^i to count how many of the cells in the i-th block in P^* are now placed in the j-th block in P.

$$B_j^i = |\mathcal{A}_i^* \cap \mathcal{A}_j|$$

3.3 Cost Function

In this section, we define a multi-objective cost function with a cut quality factor and a similarity factor. We associate the original K^*-way partition P^* with the original hypergraph $H^* = (\mathcal{C}^*, \mathcal{N}^*)$ and a current K-way partition P with a modified hypergraph $H = (\mathcal{C}, \mathcal{N})$ in the following discussion. Note that K is not necessarily the same as K^*.

Definition 3.6. For a given hypergraph H, the *partition cost* of P consists of the *cut cost*, which corresponds to the number of interconnection between blocks, and the *similarity cost* which measures the degree of similarity to the original partition

P^*. As a result, the partition cost of P is defined as

$$f_{par}(P) = f_{cut}(P) + Rf_{sim}(P)$$

where the *similarity coefficient* R is a positive weighting constant for similarity cost.

Definition 3.7. Consider a cell $C \in \mathcal{A}_i^*$ in the original partition P^*. *Similarity cost* of the cell C in block $\mathcal{A}_j$ in the current partition P is defined as the number of *missing block neighbors* of C that were placed in the same block in P^* but now are in a different block in P. The newly added cells into H which did not exist in H^* are assigned a similarity cost of 0. For a cell $C \in \mathcal{A}_j \subset \mathcal{C}$,

$$f_{sim}(C) = \begin{cases} |\overline{BN_{\mathcal{A}_j}}(C)| = \sum_{k \neq j} B_k^i & \text{if } C \in \mathcal{A}_i^* \subset \mathcal{C}^* \\ 0 & \text{if } C \notin \mathcal{C}^* \end{cases}$$

The similarity cost of an original block $\mathcal{A}_i^*$ is defined as

$$\begin{aligned} f_{sim}(\mathcal{A}_i^*) &= \sum_{C \in \mathcal{A}_i^* \cap \mathcal{C}} f_{sim}(C) \\ &= \sum_j \sum_{C \in \mathcal{A}_i^* \cap \mathcal{A}_j} |\overline{BN_{\mathcal{A}_j}}(C)| \\ &= \sum_j (B_j^i \sum_{k \neq j} B_k^i) \end{aligned}$$

The similarity cost of partition P with respect to P^* is now defined as

$$\begin{aligned} f_{sim}(P) &= \sum_{C \in \mathcal{C}} f_{sim}(C) \\ &= \sum_i f_{sim}(\mathcal{A}_i^*) \\ &= \sum_i \sum_j \sum_{k \neq j} B_j^i B_k^i \\ &= 2 \sum_i \sum_j \sum_{k > j} B_j^i B_k^i \end{aligned}$$

The normalized similarity cost is defined as $f_{sim}(P)/|\mathcal{C}|$, which is interpreted as the average number of missing block neighbors per cell.

If the cells are more aggregated as in the original partition, the similarity cost is lower, whereas if the cells in a same block are more scattered in the current partition, similarity cost is higher.[1] For instance, if two hypergraph H^* and H are identical, it is obvious that $f_{sim}(P) = 0$ when $P = P^*$. Note that if there is only a change in the ordering of blocks, it still is regarded as the same partition, hence $f_{sim}(P) = 0$. Figure 3.2 shows examples of the proposed similarity cost.

Definition 3.8. For any cell C such that $C \in \mathcal{A}_j \subset \mathcal{C}$ in P and $C \in \mathcal{A}_i^* \subset \mathcal{C}^*$ in P^*, if there are no missing block neighbors of the cell C in $\mathcal{A}_j$ then the partition P is said to be a *perfectly preserved partition* of P^*, and is denoted by $P \preceq P^*$.

$$P \preceq P^* \equiv \forall C \in \mathcal{A}_j \subset \mathcal{C}, |\overline{BN_{\mathcal{A}_j}}(C)| = \sum_{k \neq j} B_k^i = 0$$

It is obvious that $P \preceq P^*$ if and only if $f_{sim}(P) = 0$ with respect to P^*.

Definition 3.9. The *gain* of a cell C in block $\mathcal{A}_i$ to block $\mathcal{A}_j$, $\gamma_{\mathcal{A}_j}(C)$, represents the amount of cost reduction after C is moved to $\mathcal{A}_j$, $1 \leq i \neq j \leq K$.

$$\gamma_{\mathcal{A}_j}(C) = f_{par}(P) - f_{par}(P')$$

where P' is a new partition generated by the move of C. Cut gain $\kappa_{\mathcal{A}_j}(C)$ and similarity gain $\lambda_{\mathcal{A}_j}(C)$ are defined respectively:

$$\begin{aligned} \kappa_{\mathcal{A}_j}(C) &= f_{cut}(P) - f_{cut}(P') \\ \lambda_{\mathcal{A}_j}(C) &= f_{sim}(P) - f_{sim}(P') \end{aligned}$$

[1] In the sense that a lower value of the similarity cost represents a lower degree of discrepancy in groupings, similarity cost, semantically, has the meaning of discrepancy cost.

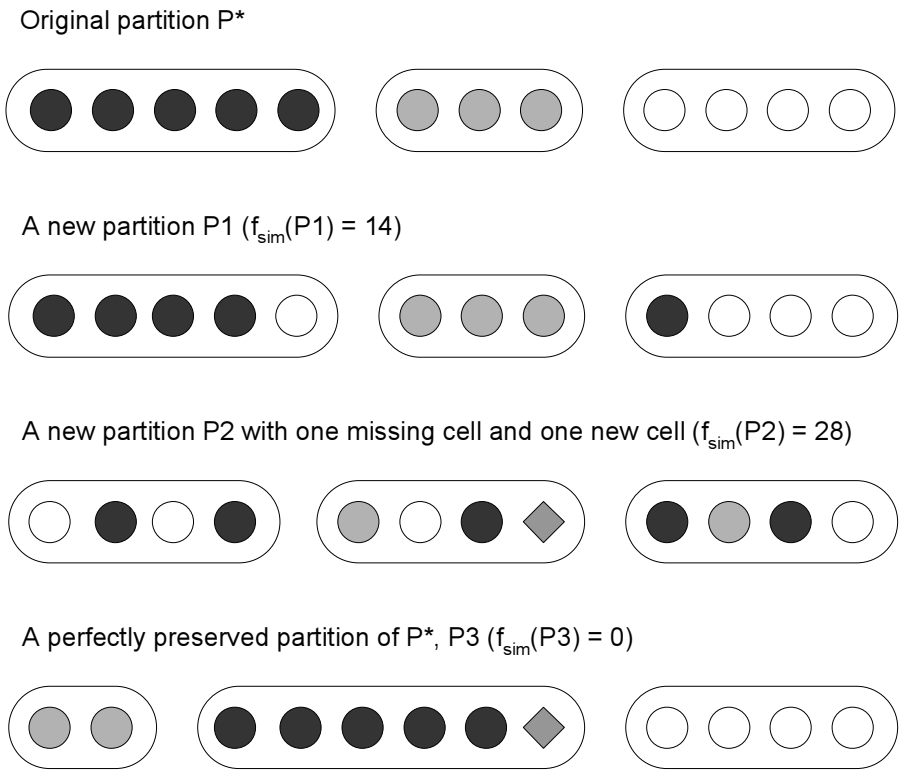


Figure 3.2: Similarity cost

Then, we have

$$\gamma_{\mathcal{A}_j}(C) = \kappa_{\mathcal{A}_j}(C) + R\lambda_{\mathcal{A}_j}(C)$$

3.4 Proposed Algorithm

We consider a current hypergraph H that is obtained from applying any of the following modifications on the original hypergraph H^*: cell resizing, cell addition, cell deletion, net addition, and net deletion. Note that even only with some cell resizing it may not be possible to keep the original partition P^* since the feasibility

Procedure `simP`

Input: a given netlist hypergraph hypergraph $H = (\mathcal{C}, \mathcal{N})$
K = number of blocks
R = similarity coefficient
P^* = K^*-way original partition on H^*
Output: K-way partition $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$

1. Construct a balanced initial partition P_i from P^* and set $P := P_i$
2. **repeat**
3. Initialize the gains for all cells based on P
4. **while** there exists a movable free cell **do**
5. Select a free cell C with the highest gain
($\mathcal{A}_i$ = source block, $\mathcal{A}_j$ = destination block)
6. Assign cell C to block $\mathcal{A}_j$
7. Lock cell C
8. Update the cut gains for the affected free cells
9. Update the similarity gains for the affected free cells
10. **end while**
11. P := the best partition with the lowest cost among
the partitions produced in above while loop
12. **until** there is no improvement in the cost of partition

Figure 3.3: Proposed algorithm (simP)

is broken due to the violated balance constraint.

3.4.1 Stable Multiway Partitioning

The basic structure of the proposed algorithm is similar to the algorithm used in [33, 57, 55] since it is based on the iterative improvement partitioning (IIP) technique. Figure 3.3 shows the structure of the proposed algorithm.

In constructing an initial partition (step 1 of Figure 3.3), there are two different possible approaches. One approach is to simply create a random initial partition, like most of the existing partitioning algorithms. In this approach, not only will the initial partition be random but the initial value of the similarity cost will also be some random value (possibly very large). Another approach that can be used to

create the initial partition is to begin with a perfectly preserved partition directly derived from the original partition P^*. In this scenario, ideally $f_{sim} = 0$ since the groupings of the existing cells are identical to their groupings in P^*. However, the balance constraint is usually violated due to the cell resizings and/or cell additions/deletions. Hence, the initial partition is forced to be balanced within a given balance constraint by a small number of random cell moves from larger blocks to smaller blocks. Though the first approach (which is called *constructive* approach since stability is constructed during the series of the cell moves in the algorithm) could be useful for some other applications, we use the second approach (which is called *destructive* approach) for ECO applications — the algorithm begins with a very stable partition but moves away from this stability to achieve a lower overall cost. We observed the constructive approach becomes effective if the modification is significantly large, but for partial modification that occurs in ECO situations, the destructive approach is suitable.

The main operations of the procedure simP include the selection of a best cell based on the cell gain (step 5) and the update of the gains of the affected cells (step 8, 9). The gain update operation must be designed efficiently because the cell gains are used in the selection of a best cell. Computation and update of cut gains are described in [33, 57].

3.4.2 Gain Computation

Since our approach is built on the conventional IIP flow [57, 33, 55], we restrict our focus to similarity gain computation, which is introduced as a new quality factor. In the following discussion, we ignore the similarity gain computation of the cells

which were newly added into the current hypergraph H, since their similarity costs and gains always remain zero throughout the process.

From Definition 3.7 and Definition 3.9, we have the following lemmas.

Lemma 3.1. Let a cell C be in $\mathcal{A}_i^*$ in the original partition P^*, and be in $\mathcal{A}_j$ in P. In a partition $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$ on a hypergraph $H = (\mathcal{C}, \mathcal{N})$, the *similarity gain* associated with moving cell C from block $\mathcal{A}_j$ to block $\mathcal{A}_k$ is

$$\lambda_{\mathcal{A}_k}(C) = 2(|BN_{\mathcal{A}_k}(C)| - |BN_{\mathcal{A}_j}(C)| + 1) = 2(B_k^i - B_j^i + 1)$$

Proof. Moving the cell C in block $\mathcal{A}_j$ to block $\mathcal{A}_k$ affects the similarity costs of the cells in $BN_{\mathcal{A}_j}(C)$ (including itself) and $BN_{\mathcal{A}_k}(C)$. Let f_{sim} and f'_{sim} be the similarity cost before and after the move of C, respectively.

$$f_{sim} = \sum_{C' \in BN(C)} f_{sim}(C') + \sum_{C' \notin BN(C)} f_{sim}(C')$$

$$f'_{sim} = \sum_{C' \in BN(C)} f'_{sim}(C') + \sum_{C' \notin BN(C)} f'_{sim}(C')$$

1) For cell C itself,

$$\begin{aligned} f_{sim}(C) - f'_{sim}(C) &= |\overline{BN_{\mathcal{A}_j}}(C)| - (|\overline{BN_{\mathcal{A}_k}}(C)| - 1) \\ &= (|BN(C)| - |BN_{\mathcal{A}_j}(C)|) - (|BN(C)| - |BN_{\mathcal{A}_k}(C)| - 1) \\ &= |BN_{\mathcal{A}_k}(C)| - |BN_{\mathcal{A}_j}(C)| + 1 \end{aligned}$$

2) For $C' \in BN_{\mathcal{A}_j}(C) - \{C\}$,

$$\begin{aligned} f_{sim}(C') - f'_{sim}(C') &= |\overline{BN_{\mathcal{A}_j}}(C')| - (|\overline{BN_{\mathcal{A}_j}}(C')| + 1) \\ &= -1 \end{aligned}$$

3) For $C' \in BN_{\mathcal{A}_k}(C)$,

$$\begin{aligned} f_{sim}(C') - f'_{sim}(C') &= |\overline{BN_{\mathcal{A}_k}}(C')| - (|\overline{BN_{\mathcal{A}_k}}(C')| - 1) \\ &= 1 \end{aligned}$$

4) For all other cells C',

$$f_{sim}(C') - f'_{sim}(C') = 0$$

Thus, the similarity gain of C for $\mathcal{A}_k$ is

$$\begin{aligned} \lambda_{\mathcal{A}_k}(C) &= f_{sim} - f'_{sim} \\ &= (|BN_{\mathcal{A}_k}(C)| - |BN_{\mathcal{A}_j}(C)| + 1) \\ &\quad +(-1)(|BN_{\mathcal{A}_j}(C)| - 1) \\ &\quad +(+1)(|BN_{\mathcal{A}_k}(C)|) \\ &= 2(|BN_{\mathcal{A}_k}(C)| - |BN_{\mathcal{A}_j}(C)| + 1) \\ &= 2(B_k^i - B_j^i + 1) \end{aligned}$$

□

Lemma 3.2. Let a cell C be in block $\mathcal{A}_i$. After C moves to block $\mathcal{A}_j$, the similarity gain of every free cell C' is updated as follows:

1) For $C' \in BN_{\mathcal{A}_i}(C)$,

$$\begin{aligned} \lambda^{new}_{\mathcal{A}_k}(C') &= \lambda^{old}_{\mathcal{A}_k}(C') + 2 \\ \lambda^{new}_{\mathcal{A}_j}(C') &= \lambda^{old}_{\mathcal{A}_j}(C') + 4 \end{aligned}$$

2) For $C' \in BN_{\mathcal{A}_j}(C)$,

$$\begin{aligned} \lambda^{new}_{\mathcal{A}_k}(C') &= \lambda^{old}_{\mathcal{A}_k}(C') - 2 \\ \lambda^{new}_{\mathcal{A}_i}(C') &= \lambda^{old}_{\mathcal{A}_i}(C') - 4 \end{aligned}$$

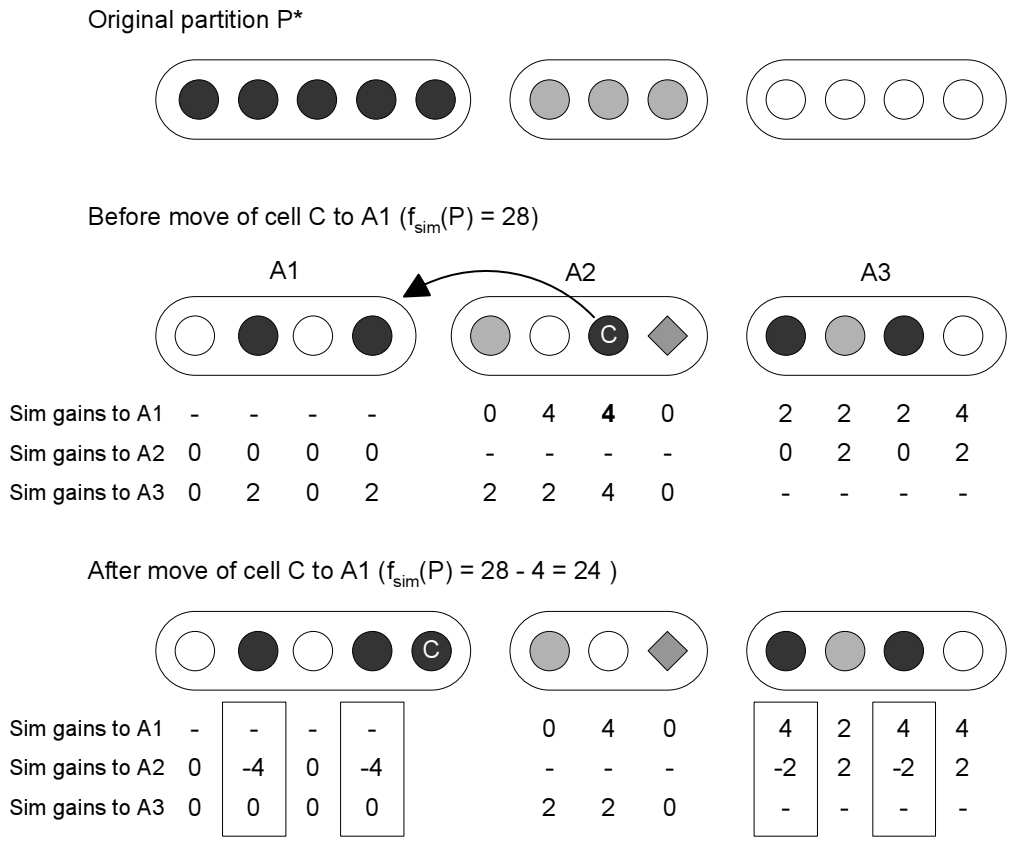


Figure 3.4: Similarity gain computation

3) For $C' \in BN_{\mathcal{A}_k}(C)$,

$$\begin{aligned} \lambda^{new}_{\mathcal{A}_i}(C') &= \lambda^{old}_{\mathcal{A}_i}(C') - 2 \\ \lambda^{new}_{\mathcal{A}_j}(C') &= \lambda^{old}_{\mathcal{A}_j}(C') + 2 \end{aligned}$$

where $1 \leq k \neq i, j \leq n$.

Proof. The similarity gain updates are required only for block neighbors of the cell C which are free.

1) For $C' \in BN_{\mathcal{A}_i}(C)$,

$$\begin{aligned}
\lambda^{new}_{\mathcal{A}_k}(C') &= 2(|BN_{\mathcal{A}_k}(C')| - (|BN_{\mathcal{A}_i}(C')| - 1) + 1) \\
&= \lambda^{old}_{\mathcal{A}_k}(C') + 2, \\
\lambda^{new}_{\mathcal{A}_j}(C') &= 2((|BN_{\mathcal{A}_j}(C')| + 1) - (|BN_{\mathcal{A}_i}(C')| - 1) + 1) \\
&= \lambda^{old}_{\mathcal{A}_j}(C') + 4
\end{aligned}$$

2) For $C' \in BN_{\mathcal{A}_j}(C)$,

$$\begin{aligned}
\lambda^{new}_{\mathcal{A}_k}(C') &= 2(|BN_{\mathcal{A}_k}(C')| - (|BN_{\mathcal{A}_j}(C')| + 1) + 1) \\
&= \lambda^{old}_{\mathcal{A}_k}(C') - 2, \\
\lambda^{new}_{\mathcal{A}_i}(C') &= 2((|BN_{\mathcal{A}_i}(C')| - 1) - (|BN_{\mathcal{A}_j}(C')| + 1) + 1) \\
&= \lambda^{old}_{\mathcal{A}_i}(C') - 4
\end{aligned}$$

3) For $C' \in BN_{\mathcal{A}_k}(C)$,

$$\begin{aligned}
\lambda^{new}_{\mathcal{A}_i}(C') &= 2((|BN_{\mathcal{A}_i}(C')| - 1) - |BN_{\mathcal{A}_k}(C')| + 1) \\
&= \lambda^{old}_{\mathcal{A}_i}(C') - 2, \\
\lambda^{new}_{\mathcal{A}_j}(C') &= 2((|BN_{\mathcal{A}_j}(C')| + 1) - |BN_{\mathcal{A}_k}(C')| + 1) \\
&= \lambda^{old}_{\mathcal{A}_j}(C') + 2
\end{aligned}$$

For all other cells $C' \notin BN(C)$, $\lambda(C')$ remain unchanged. □

Figure 3.4 shows an example of the gain computations associated with a cell move.

3.4.3 Similarity Coefficient

The similarity coefficient R in the cost function plays an important role in the performance of the proposed algorithm, since it determines the degree of relative emphasis on similarity compared to cut quality. Adjusting the similarity coefficient gives options for desired performance with a trade-off of cut quality and similarity. The observations to find reasonable values for R are presented in the following lemmas.

Lemma 3.3. During the execution of the proposed algorithm, the range of similarity gain of any cell C is bounded by

$$-2(M^* - 1) \leq \lambda_{\mathcal{A}_k}(C) \leq 2(M^* - 1),$$

where $C \in \mathcal{A}_j$, $k \neq j$, $M^* = max_{1 \leq i \leq K^*} |\mathcal{A}_i^*|$.

Proof. Let a cell C be in block $\mathcal{A}_i^*$ in P^*, and now be in block $\mathcal{A}_j$ in P. From Lemma 3.1 and, $0 \leq |BN_{\mathcal{A}_k}(C)| \leq |\mathcal{A}_i^*| - 1$ and $1 \leq |BN_{\mathcal{A}_j}(C)| \leq |\mathcal{A}_i^*|$, we have

$$-2(|\mathcal{A}_i^*| - 1) \leq \lambda_{\mathcal{A}_k}(C) \leq 2(|\mathcal{A}_i^*| - 1)$$

From $M^* = max_i |\mathcal{A}_i^*|$,

$$-2(M^* - 1) \leq \lambda_{\mathcal{A}_k}(C) \leq 2(M^* - 1) \qquad \square$$

Theorem 3.1. Let a balance ratio be sufficiently large such that all the cells with the highest gain do not violate the balance constraint. Starting from any arbitrary initial partition, the proposed algorithm always returns a perfectly preserved partition of P^*, if $R > p/2$.

Proof. Let the resulting partition from the algorithm be $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$. Then, the value of the gain(γ) of each cell under the resulting partition must be *nonpositive*. Suppose that P is not a perfectly preserved partition of P^*, i.e., $f_{sim}(P) > 0$. Then, there exist some cells with positive similarity gains. Let C be such a cell in $\mathcal{A}_j$ with positive similarity gain so that $2 \leq \lambda_{\mathcal{A}_k}(C) \leq 2(M^* - 1)$ where $k \neq j$. For the cell C, $\gamma_{\mathcal{A}_k}(C)$ is as follows:

$$\gamma_{\mathcal{A}_k}(C) = \kappa_{\mathcal{A}_k}(C) + R\lambda_{\mathcal{A}_k}(C)$$

Since $-p \leq \kappa_{\mathcal{A}_k}(C) \leq p$ and $R\lambda_{\mathcal{A}_k}(C) > (p/2)2 = p$, $\gamma_{\mathcal{A}_k}(C) > 0$ for all those cells with positive similarity gains, which is a contradiction to that $\gamma_{\mathcal{A}_k}(C)$ in resulting partition P is nonpositive. Therefore, $P \preceq P^*$ and $f_{sim}(P) = 0$ with respect to P^*. □

Lemma 3.4. For a cell C, the range of the similarity gain weighted by R is the same as the range of the cut gain, if $R = p/(2(M^* - 1))$.

Proof. It follows from Lemma 3.3 and $-p \leq \kappa_{\mathcal{A}_k}(C) \leq p$ for any cell C in $\mathcal{A}_j$ $(k \neq j)$. □

Based on the observation in Lemma 3.4, we use $R^* = p_{avg}/(2(M^* - 1))$ as a basis value of the similarity coefficient in our experiments.

3.5 Extension to Multilevel Paradigm

Recently, a new multilevel partitioning paradigm has been introduced, which is capable of finding higher quality solutions than flat or bi-level partitioners in dramatically reduced run time with the two-digit times speed-up [2, 39, 41]. The multilevel

partitioning consists of three phases: multilevel coarsening, initial partitioning at the coarsest level, and multilevel uncoarsening with FM refinement. During the coarsening phase, the problem size is gradually reduced over the levels while capturing strong connectivities in the circuit netlist. At the coarsest level, a relatively high quality initial partitioning solution is quickly obtained. Then, the current partitioning solution is successively propagated to lower levels, where the partitioning solution on the bigger problem keeps improving by FM refinement.

The use of multi-phase refinement with restricted coarsening (clustering), namely V-cycle, has been suggested as a postprocessing procedure of the multilevel partitioning [39, 40]. The V-cycle is an incremental multilevel partitioning with a *restricted* coarsening phase and an uncoarsening/refinement phase. During the restricted coarsening phase, the multilevel clustering always preserves the previous partition P^* such that any clusters which belong to different blocks in P^* are not allowed to merge together. This clustering scope restriction and a different randomization seed cause the resulting multilevel clustering tree to differ from the tree constructed while previously producing P^*. However, at the coarsest level, the underlying partitioning result is exactly identical to P^*, and while traversing down to lower (finer) levels, P^* is further refined by FM partitioning. The V-cycle is used as an post-attempt to improve the cut quality of the current solution with the assumption that no modifications have been made to the original netlist hypergraph. Note that V-cycle does not provide a way to accommodate the netlist hypergraph modification, nor is capable of quantifying the similarity of the current partition with respect to the previous partition.

3.5.1 Restricted Coarsening

Based on the proposed cost functions and the structure of V-cycle, we extend and incorporate the proposed algorithm into the multilevel paradigm, as follows. Since the proposed algorithm is also an incremental partitioning, the skeleton of our multilevel implementation is similar to that of V-cycle (See Figure 3.5); we also follow the concept of 'restricted coarsening' to create an initial partition at the coarsest level, with the exception that the newly added cells (introduced by the modification of the netlist hypergraph) have freedom to be clustered with other clusters. Once such a new cell (or a cluster consisting of new cells only) is merged with another cluster that belongs to block $\mathcal{A}_i$, the resulting cluster is regarded to belong to $\mathcal{A}_i$. After the block membership of the cluster that includes one or more newly added leaf cells is determined, the cluster is never to be clustered with other clusters with different block memberships, at any levels of clustering. In this fashion, it is guaranteed that the partition P_l at any current level l is a perfectly preserved partition of P^* (i.e., $f_{sim}(P_l) = 0, 0 \leq l \leq t$, where t is the coarsest top level). Note that if there are no newly added cells, the coarsening phase is identical to that of V-cycle. Unlike V-cycle, the feasibility of the preliminary initial partition P_t at the top level t is not guaranteed, due to the cell size changes as well as the presence of newly added cells. Hence, the actual initial partition $(P_t)_i$ is obtained by forcing the balance constraints on the current (perfectly preserved) partition P_t, yielding some positive similarity cost. The balance is forced by randomly moving some clusters in oversized blocks to undersized blocks until the balance constraint is satisfied.

Procedure `simPml`

Input: a given netlist hypergraph hypergraph $H = (\mathcal{C}, \mathcal{N})$
$P^* = K^*$-way original partition on H^*
Output: K-way partition $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$

1. Construct a t-level clustering tree from H using restricted coarsening based on P^*
2. Obtain a balanced initial partition $(P_t)_i$ on H_t from $P_t \preceq P^*$
3. $P_t :=$ `simP`$(H_t, (P_t)_i)$
4. **for** $l = t - 1$ down to 0 **do**
5. Project H_{l+1} onto H_l
6. $P_l :=$ `simP`(H_l, P_{l+1})

Figure 3.5: Outline of the multilevel implementation (simPml)

3.5.2 Refinement with Similarity Cost

The hypergraph uncoarsening and the projection of the current best partition onto the lower levels remains the same as those of conventional multilevel partitioning approach. However, the FM refinement at each level is replaced by the partitioning with multiple objective costs —cut quality cost and similarity cost— discussed in the previous sections. In particular, the similarity gain computation and the similarity coefficient are two major issues to be addressed in the multilevel implementation.

Thanks to the restricted coarsening, every cluster at any level in the multilevel clustering tree is *homogeneous*, i.e., any cells that belong to different blocks in the previous partition P^* are not located together in the same cluster even though some newly added cells (not affecting the similarity cost) can be grouped with the cells with the same block membership, solely by the degree of connectivities. Hence, Lemma 3.1 is easily generalized as follows.

Corollary 3.1. Let a homogeneous cluster C^+ have x cells in $\mathcal{A}_i^*$ in the original partition P^*, and have $|C^+| - x$ newly added cells, where $0 \leq x \leq |C^+|$. Also let the cluster C^+ be a subset of $\mathcal{A}_j$ in the current partition P, then, in a partition

$P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_K)$ on a hypergraph $H = (\mathcal{C}, \mathcal{N})$, the *similarity gain* associated with moving cluster C^+ from block $\mathcal{A}_j$ to block $\mathcal{A}_k$ is

$$\lambda_{\mathcal{A}_k}(C^+) = 2x(|BN_{\mathcal{A}_k}(C^+)| - |BN_{\mathcal{A}_j}(C^+)| + x) = 2x(B_k^i - B_j^i + x)$$

where $x = |C^+ \cap \mathcal{A}_i^*|$, and $BN_{\mathcal{A}_k}(C^+)$ and $BN_{\mathcal{A}_j}(C^+)$ are defined as $BN_{\mathcal{A}_k}(C)$ and $BN_{\mathcal{A}_j}(C)$, respectively, for any cell C of x cells such that $C \in C^+ \cap \mathcal{A}_i^*$. Note that, for any $C_1, C_2 \in C^+ \cap \mathcal{A}_i^*$, $BN_{\mathcal{A}_j}(C_1) = BN_{\mathcal{A}_j}(C_2)$ and $BN_{\mathcal{A}_k}(C_1) = BN_{\mathcal{A}_k}(C_2)$.

Proof. It directly follows from the extension of the proof of Lemma 3.1 to the moves of x cells. □

In a multilevel implementation, one move of a cluster at a higher (coarser) level is equivalent to the moves of the significantly many leaf cells. As implied in Corollary 3.1, the group migration caused by the single move of a cluster with x existing (non-new) cells causes the higher degree of increase/decrease of the similarity cost at once; actually the amount of similarity cost change is about x times more than that in a single leaf cell migration. However, the cut cost change associated with a cluster move is not proportional to the number of non-new leaf cells, x. For the similarity cost and cut cost to fairly compete with each other, the similarity coefficient needs to be adjusted based on the size of the cluster; hence we redefine the partition cost and the gain of a cluster, where the emphasis on the similarity cost is intentionally downscaled according to the size of each cluster.

Definition 3.10. For a hypergraph $H_l = (\mathcal{C}_l, \mathcal{N}_l)$ at level l in the multilevel clustering tree, the similarity cost of a cluster $C^+ \in \mathcal{C}_l$ is defined as

$$f_{sim}(C^+) = \sum_{C \in C^+} f_{sim}(C)$$

Note that, if C^+ is homogeneous, the similarity costs of any non-new cells in C^+ are identical. Let x be a function defined on the set of homogeneous clusters C^+ in $\mathcal{C}_l$ that returns the number of non-new leaf cells which have the same block membership in P^*. The *partition cost* of P_l is defined as

$$f_{par}(P_l) = f_{cut}(P_l) + \sum_{C^+ \in \mathcal{C}_l} R/x(C^+) \cdot f_{sim}(C^+)$$

where the R is a similarity coefficient given at the lowest level hypergraph H_0.

Accordingly, the *gain* of a cluster C^+ in block $\mathcal{A}_i$ to block $\mathcal{A}_j$ is computed as

$$\gamma_{\mathcal{A}_j}(C^+) = \kappa_{\mathcal{A}_j}(C^+) + R/x(C^+) \cdot \lambda_{\mathcal{A}_j}(C^+)$$

3.6 Experimental Results

We implemented our algorithm in two versions —flat version(`simP`) and multilevel version(`simPml`)— in C++/STL, and evaluated the performance on nine ISPD benchmark circuits [1]. The flat version exactly follows the proposed algorithm described in Figure 3.3, and the multilevel version uses `simP` as the underlying partitioner on the multilevel paradigm. We first provide our experiment setup for both implementations; then, the results from the flat version, `simP`, are discussed focusing on the trade-off between cut quality and stability. Finally, the results from the multilevel version, `simPml`, are compared to those from `simP` to evaluate functionality and speed-up.

For each circuit, a *golden* partition P^* has been obtained by taking the best result from 100 runs of K-way hMetis. Then, we have injected some impurities

such as cell resizing and addition/deletion of cells/nets into the original netlist hypergraph H^* to simulate netlist modifications. With the knowledge of the previous partitioning result, the modified netlist H is partitioned by the proposed algorithm with different values of the similarity coefficient, and the quality is compared with the fresh runs of hMetis on H. Note that our algorithm was applied only once, but hMetis performed 10 runs to obtain a well-optimized solution in terms of cut quality.

Table 3.1, 3.2, and 3.3 shows the cut costs and similarity costs of the 8-way partitions on the modified netlists under different modification scenarios. The column (a) and (b) represent the golden partitions on the original netlists and the initial partitions directly derived from the golden partitions, respectively. An initial (feasible) partition P_i is obtained just by forcing the balance constraint on a perfectly preserved partition of the original partition. The column (c) and (d) shows the results of the proposed approach as the similarity coefficient varies from 0 to R^*. In (c) similarity is totally ignored and only cut cost minimization is targeted starting from P_i, while the increase of R gradually emphasizes the similarity. The last column (e) shows the best results from fresh 10 runs of hMetis on the modified netlists in terms of cut quality.

It is shown that, as the value of R increases, we have more similar(stable) partitioning solutions with a little sacrifice of the cut quality (See Figure 3.6). Thanks to the high qualities of the golden partitions, only one run of `simP` yields, in many cases, better solutions than the best of 10 runs of hMetis (bold-faced in the tables). The hMetis produces radically different partitions from the golden partitions with enormously high similarity costs, however the solutions from `simP` are quite sta-

Circuit		(a)	(b)	(c)	(d)							(e)
		P^*	P_i	$0R^*$	$0.005R^*$	$0.01R^*$	$0.025R^*$	$0.05R^*$	$0.1R^*$	$0.25R^*$	$1R^*$	hMetis
ibm01	cut	978	1093	972	975	978	977	1018	1018	1020	1049	931
	n. sim	0	5.1	24.8	16.1	18.1	11.1	2.7	2.7	2.7	3.0	460.9
ibm03	cut	3329	3729	**3308**	**3310**	**3317**	**3362**	3414	3520	3525	3546	3381
	n. sim	0	25.1	90.6	59.2	54.8	47.9	46.4	25.4	23.9	23.7	905.6
ibm05	cut	6494	6755	**6323**	**6330**	**6336**	**6414**	**6414**	**6525**	**6528**	**6633**	6866
	n. sim	0	18.4	353.7	100.8	104.2	47.5	47.5	19.0	18.4	17.1	1522.8
ibm07	cut	4126	4546	**4123**	**4246**	**4233**	**4325**	**4434**	**4434**	**4434**	4477	4454
	n. sim	0	28.9	112.7	34.6	37.0	34.7	28.5	28.5	28.5	28.0	841.9
ibm09	cut	3269	3981	**3237**	**3310**	**3298**	3383	**3326**	3340	3391	3602	3337
	n. sim	0	43.7	93.3	53.9	53.7	56.6	33.7	31.9	25.4	27.6	1727.3
ibm11	cut	4183	4499	**4135**	**4158**	**4162**	**4162**	**4162**	**4182**	**4187**	4344	4304
	n. sim	0	22.0	103.3	29.8	28.9	29.2	29.2	14.4	12.6	16.5	2641.1
ibm13	cut	3604	3964	**3556**	**3627**	**3624**	**3669**	**3669**	**3669**	3700	3801	3682
	n. sim	0	21.2	289.5	45.0	24.5	8.5	10.2	8.5	10.4	12.1	3712.3
ibm15	cut	8558	8875	**8531**	**8508**	**8528**	8857	8857	8857	8858	8858	8614
	n. sim	0	18.9	152.5	58.8	53.1	18.9	18.9	18.9	18.6	18.6	7038.4
ibm17	cut	12237	12237	**12032**	**12058**	**12056**	**12182**	**12236**	**12236**	**12237**	**12237**	12955
	n. sim	0	0	259.8	71.9	71.9	15.6	0.2	0.2	0	0	9072.0

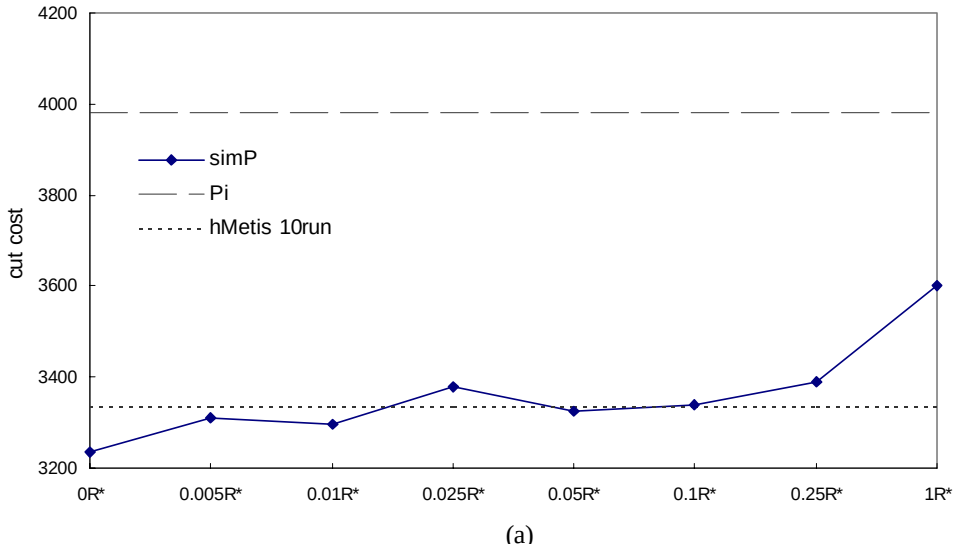
Table 3.1: `simP` results: Cut cost and normalized similarity cost for 8-way partitioning with 5% balance ratio. 10% of the original cells have been resized to $\times 2$, $\times 4$ or $\times 8$. (a) the original golden partition from 100 runs of hMetis on H^*, (b) the initial partition on H derived from P^*, (c) the resulting partition from P_i without similarity consideration, (d) the resulting partition from P_i with a varying similarity coefficient, (e) the resulting partition by 10 runs of hMetis

Circuit		(a)	(b)	(c)	(d)							(e)
		P^*	P_i	$0R^*$	$0.005R^*$	$0.01R^*$	$0.025R^*$	$0.05R^*$	$0.1R^*$	$0.25R^*$	$1R^*$	hMetis
ibm01	cut	978	1077	**945**	**956**	**954**	999	1001	999	1008	1019	956
	n. sim	0	3.6	81.4	16.8	14.6	4.9	5.3	5.3	5.4	3.0	362.8
ibm03	cut	3329	3410	**3291**	**3290**	**3298**	**3301**	**3325**	**3325**	**3328**	**3341**	3427
	n. sim	0	4.1	162.6	23.1	13.5	9.9	7.3	7.3	4.8	1.6	780.3
ibm05	cut	6494	6494	**6320**	**6337**	**6353**	**6454**	**6454**	**6454**	**6454**	**6482**	6898
	n. sim	0	0	333.8	87.9	71.6	6.3	6.3	6.3	6.3	0.7	1569.1
ibm07	cut	4126	5486	**4182**	**4254**	**4328**	**4279**	**4307**	4553	4724	4761	4468
	n. sim	0	92.6	125.3	73.7	74.5	75.5	85.7	62.8	68.1	62.6	1157.4
ibm09	cut	3269	4652	3319	3404	3317	3405	3330	**3304**	3884	4144	3307
	n. sim	0	88.4	178.9	95.6	61.0	70.0	51.9	61.9	58.0	62.9	1699.3
ibm11	cut	4183	4533	**4152**	**4154**	**4152**	**4190**	**4193**	**4193**	**4230**	4288	4257
	n. sim	0	23.7	299.2	50.9	34.6	31.0	29.7	32.4	17.4	18.5	1348.2
ibm13	cut	3604	3604	3552	3573	3604	3604	3604	3604	3604	3604	3543
	n. sim	0	0	135.9	31.6	0	0	0	0	0	0	2311.8
ibm15	cut	8558	8876	**8529**	**8518**	**8546**	**8560**	8842	8842	8844	8859	8820
	n. sim	0	16.5	159.8	60.6	43.7	52.9	17.2	17.2	16.7	16.2	9727.9
ibm17	cut	12237	12237	**12032**	**12045**	**12057**	**12182**	**12236**	**12236**	**12237**	**12237**	12261
	n. sim	0	0	260.0	73.0	67.7	15.6	0.2	0.2	0	0	8275.5

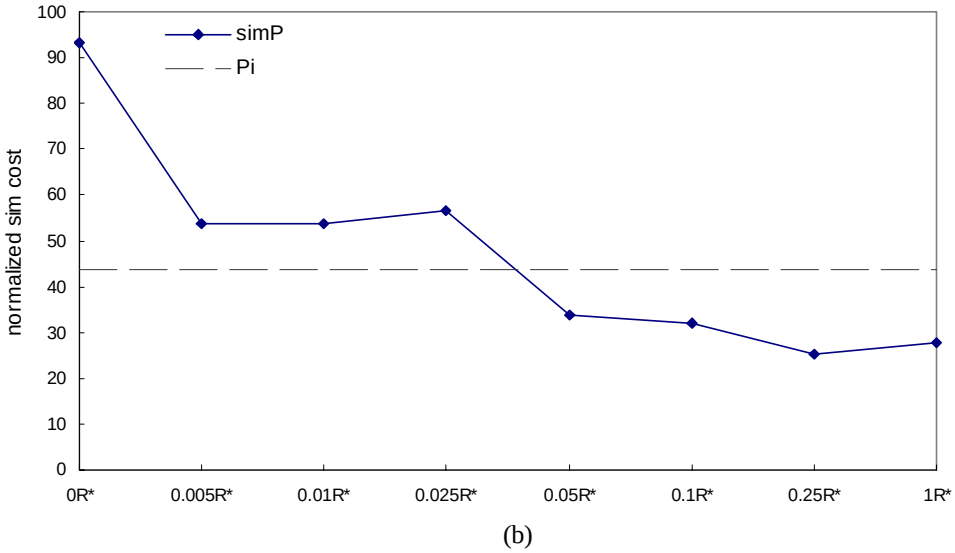
Table 3.2: `simP` results: Cut cost and normalized similarity cost for 8-way partitioning with 5% balance ratio. 20% of the original cells have been resized to ×2, ×4 or ×8. (a),(b),(c),(d), and (e) are the same as in Table 3.1.

Circuit		(a)	(b)	(c)	(d)							(e)
		P^*	P_i	$0R^*$	$0.005R^*$	$0.01R^*$	$0.025R^*$	$0.05R^*$	$0.1R^*$	$0.25R^*$	$1R^*$	hMetis
ibm01	cut	978	3146	**2216**	**2211**	**2211**	**2233**	**2231**	**2233**	**2232**	**2260**	2309
	n. sim	0	14.7	109.5	15.8	15.5	5.9	6.2	5.9	5.9	0.2	354.3
ibm03	cut	3329	7306	**5757**	**5760**	**5787**	**5796**	**5814**	**5816**	**5816**	**5904**	5952
	n. sim	0	9.4	115.8	44.3	30.9	24.6	9.3	8.9	8.9	0.2	845.9
ibm05	cut	6494	12054	**9241**	**9290**	**9323**	**9425**	**9515**	**9523**	**9540**	**9624**	10194
	n. sim	0	40.4	472.6	151.9	124.5	63.0	28.8	28.9	21.4	4.0	1654.1
ibm07	cut	4126	11563	**8389**	**8413**	**8437**	**8437**	**8483**	**8483**	**8489**	**8525**	8763
	n. sim	0	41.0	344.9	53.8	39.9	40.9	12.1	11.6	11.7	0.2	1092.1
ibm09	cut	3269	12334	9108	9106	9127	9169	9173	9175	9173	9232	9081
	n. sim	0	5.8	263.7	51.1	40.3	17.4	14.4	13.9	14.4	0	1750.9
ibm11	cut	4183	15811	**11444**	**11460**	**11496**	**11516**	**11517**	**11529**	**11529**	**11593**	11796
	n. sim	0	20.6	393.4	59.1	32.5	20.4	19.7	13.6	13.6	0	1631.7
ibm13	cut	3604	17872	**13033**	**13029**	**13047**	13121	13121	13121	13127	13183	13053
	n. sim	0	0	194.8	75.3	66.0	14.9	14.9	14.9	13.3	0	2663.3
ibm15	cut	8558	37380	**27871**	**27882**	**27939**	**27942**	**27963**	**27963**	**27967**	**28033**	28545
	n. sim	0	0	539.7	62.1	29.3	27.6	16.7	16.7	15.5	0	9152.2
ibm17	cut	12237	46580	**34964**	**35072**	**35080**	**35440**	**35438**	**35442**	**35497**	**35607**	36057
	n. sim	0	0	497.0	169.3	164.1	36.7	36.7	35.8	21.5	0	9063.7

Table 3.3: `simP` results: Cut cost and normalized similarity cost for 8-way partitioning with 5% balance ratio. 10% of the original cells resized to $\times 2$, $\times 4$ or $\times 8$, 5% of the cells (with the nets connecting these cells) deleted, and 5% of the cells added (with the nets connecting these cells). (a),(b),(c),(d), and (e) are the same as in Table 3.1.

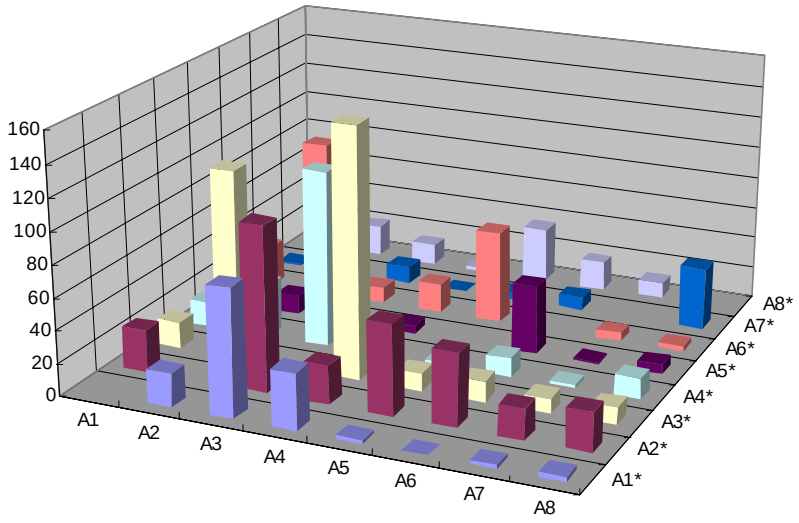


(a)

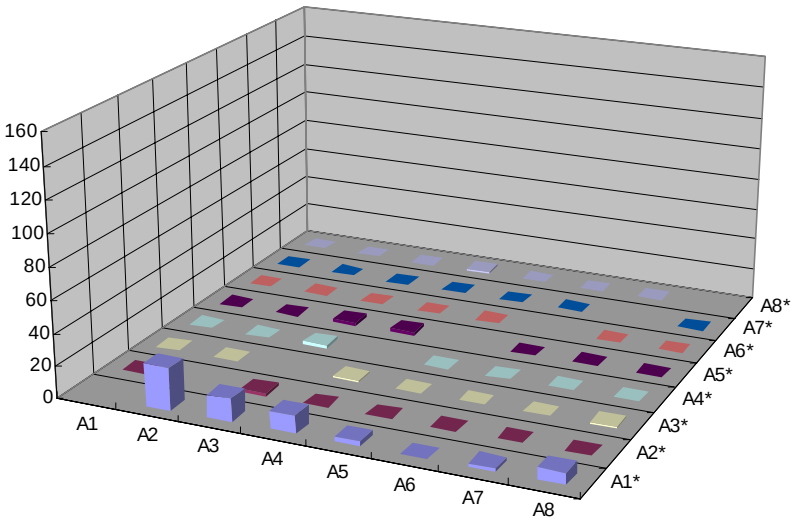


(b)

Figure 3.6: Cut-quality/stability trade-off with respect to varying R (ibm09). The similarity costs are normalized to $f_{sim}/|\mathcal{C}|$, i.e., average number of missing block neighbors per cell.



(a)



(b)

Figure 3.7: Distribution of the cells after repartitioned by simP (ibm05). Only the cells that are placed in a different block from the original block are shown. (a)$R = 0$ ($f_{cut} = 6323$, normalized $f_{sim} = 353.7$) (b)$R = 0.25R^*$ ($f_{cut} = 6528$, normalized $f_{sim} = 18.4$)

ble with respect to the original partitions, as smaller similarity costs reflecting the higher degrees of stability. Figure 3.7 visualizes how much portions of the previous partition are preserved by `simP` with different values of R. In this example, with the sacrifice of 3% cut cost increase which is still lower than the best of 10 runs of hMetis, the average number of missing block neighbors per cell decreases to 1/20, compared to the case we simply apply FM starting from P_i.

As noticed in the experimental results, some partitions produced by `simP` with higher values of similarity coefficient R are perfectly preserved partitions of the golden partitions. In these cases, the given balance constraint allows the unmodified or resized cells to remain in the previous blocks in the golden partitions, and only newly added cells make moves to improve cut quality. However, with a little perturbation by imposing lower values of R, even better cut quality solutions are obtained, which are not achievable from the fresh 10 runs of hMetis.

We observed the number of passes within one run of `simP` is affected by the value R, i.e., the more we put an emphasis on stability the fewer number of passes (65% on an average) is needed to complete the run. Another observation we made is the difficulty of balancing; if a netlist have several huge cells such as macros, each of which covers more than 10% of total area, we found that it is difficult to find a good incremental solution since many cell moves are blocked by a hard balance constraint. In order to smooth down this hard constraint, we will investigate further on the feasibility of the use of balance as a cost, not as a constraint.

The results from the multilevel implementation, `simPml`, is shown in Table 3.4, where the modification scenario is the same as the setup in Table 3.3. In general, `simPml` shows the same trend of the cut quality/stability trade-off as that of

`simP`; however, due to the impact of the restricted multilevel coarsening phase, the higher stability of the very initial partitioing solutions (represented by the smaller similarity costs for $(P_t)_i$) was achieved at the top level. While the newly added cells have full freedom to match with others solely by their connectivities, they are positioned well in the partitioned multilevel clustering tree even before the uncoarsening/refinement phase. Also, the clustering tree, which is hierarchically constructed based on the previous partition P^*, naturally *biases* the partitioning solutions at any levels throughout the uncoarsening/refinement phase while traversing down to the bottommost level. This additional biasing effect induced by the structure of the clustering tree renders the resulting partitions less sensitive to the changes of R value, compared to the flat version's case. `simPml`, however, shows slightly worse cut quality than `simP`. For instance of $R = 0$, `simPml` leads the similarity cost to about half of `simP`'s at the expense of 1.13% increase of the cut cost averagely. The multilevel implementation drastically outperforms the flat implementation in terms of run time, showing 5 to 22 times speed-up.

3.7 Conclusion

In this chapter, we presented a new incremental partitioning algorithm. Stability is defined as an additional quality of a partitioning solution, and this meta-data is formulated as a quality factor and incorporated with cut quality to constitute a multi-objective cost function. For a partially modified netlist hypergraph and a previous partitioning solution of the original netlist, the proposed algorithm produces a similar partition to the previous partitioning solution while the cut quality of the resulting partition is superior (or comparable) to fresh multiple runs of the state-of-

Circuit		(a)	(b)	(c)	(d)							(e)
		P^*	$(P_t)_i$	$0R^*$	$0.005R^*$	$0.01R^*$	$0.025R^*$	$0.05R^*$	$0.1R^*$	$0.25R^*$	$1R^*$	hMetis
ibm01	cut	978	2544	**2232**	**2221**	**2221**	**2225**	**2243**	**2251**	**2255**	**2295**	2309
	n. sim	0	19.2	32.8	22.8	22.8	24.8	19.9	13.8	5.45	1.5	354.2
ibm03	cut	3329	6336	**5746**	**5749**	**5746**	**5771**	**5773**	**5802**	**5851**	**5907**	5952
	n. sim	0	27.9	96.4	81.7	82.1	38.3	35.3	23.1	11.4	1.53	845.9
ibm05	cut	6494	10318	**9515**	**9525**	**9519**	**9557**	**9554**	**9662**	**9708**	**983**	10194
	n. sim	0	41.5	206	164	163	153	117	66	46.6	24.5	1654.1
ibm07	cut	4126	9635	**8644**	**8627**	**8623**	**8623**	**8645**	**8648**	**8668**	**8746**	8763
	n. sim	0	127.3	136	99.6	77	77.3	56.3	43.6	31.4	14.9	1092.1
ibm09	cut	3269	10090	9148	9138	9151	9139	9163	9183	9198	9273	9081
	n. sim	0	0	70.1	53	54.7	53.9	42.2	28	18.3	0	1750.9
ibm11	cut	4183	12764	**11570**	**11578**	**11578**	**11589**	**11571**	**11612**	**11648**	11844	11796
	n. sim	0	78	184	141	141	105	86.5	52.8	43.4	21.6	1631.7
ibm13	cut	3604	14602	13154	13151	13145	13151	13165	13174	13184	13229	13053
	n. sim	0	54.7	127	88.6	85.4	88.6	68.4	60.6	11.4	0	2663.3
ibm15	cut	8558	30879	**28021**	**28011**	**28011**	**28063**	**28066**	**28098**	**28089**	**28157**	28545
	n. sim	0	104	335	312	312	152	108	93.9	16.3	0.441	9152.2
ibm17	cut	12237	39104	**35382**	**35345**	**35331**	**35439**	**35431**	**35458**	**35671**	**35799**	36057
	n. sim	0	0	344	260	250	185	133	105	30.6	0	9063.7

Table 3.4: `simPml` results: Cut cost and normalized similarity cost for 8-way partitioning with 5% balance ratio. 10% of the original cells resized to $\times 2$, $\times 4$ or $\times 8$, 5% of the cells (with the nets connecting these cells) deleted, and 5% of the cells added (with the nets connecting these cells). (a),(c),(d), and (e) are the same as in Table 3.1, (b) the balanced initial partition on H_t at the top level t derived from P^*.

the-art partitioner, hMetis. The trade-off between similarity and cut quality with respect to a varying similarity coefficient is observed in the experimental results. Furthermore, the algorithm has been incorporated into the multilevel paradigm to produce the comparable results in enormously reduced run times. The proposed algorithm is beneficial to ECO applications, since our approach helps block-level ECO placers maximize the incremental capability by minimizing the portions to be re-placed.

Chapter 4

Crowdedness-Balanced Multilevel Partitioning for Uniform Resource Utilization

In this chapter, we propose a new multi-objective multilevel K-way partitioning which is aware of resource utilization distribution, assuming the resource utilization for a partitioned block is proportional to the logic occupation and the interconnections required for the block. A new quality of the partitioning solution, *crowdedness*, is defined as a virtual complexity metric where the physical size and the local connectivity of a partitioned block are considered simultaneously in the form of a weighted sum. The partitioning solutions driven by overall cut quality minimization tend to have wide variances of local interconnections for different blocks. The difference of block sizes, combining with the variance of the interconnections, potentially leads to the significant imbalance of the crowdedness (equivalently, resource utilization),

even though the feasibility imposed by a block-size constraint is satisfied.

Using the crowdedness metric, we explore the new partitioning solution space where the local interconnections are adaptively adjusted according to the block sizes, still under the same objective of overall interconnections minimization. By the carefully designed prioritized cell move policy, the proposed crowdedness-based partitioning achieves near-optimal solutions in terms of resource utilization distribution, while the overall interconnection quality also is improved but the feasibility is barely violated. The proposed approach is practically beneficial to multi-FPGA applications, in which excessive interconnections for a FPGA generate additional logics inside of the FPGA. We also discuss the partitioning applications using a user-customizable resource priority for each block; a user can specify different relative costs of interconnection resources for different blocks.

4.1 Introduction

Given a set of cells (circuit elements) and the nets (hyperedges) connecting these cells, circuit partitioning is to partition these cells into several disjoint blocks (subcircuits). The main objective of circuit partitioning is to minimize the interconnections between the blocks. Metrics to quantify cut quality such as cut size and sum of external degree (SOED) are used as objective cost functions[4, 33]. In most cases, balanced solutions are preferred; hence, a balance constraint is given to the problem so that the resulting block sizes may not exceed the specified range. In addition to the minimization of the overall interconnections between the blocks, how the interconnections are distributed over the partitioned area is also an important issue. For instance, a certain block that requires excessively many interconnections to other

blocks may yield a resource over-utilization, called *congestion*, which originates from the discrepancy between routing demand and routing supply. However, as pointed out in [60], the partitioning solutions (or the partitioning-based placement solutions) suffer from wide variances of external degrees of the partitioned blocks (block pin counts). The distribution of individual interconnections are significantly unbalanced over the blocks, even though the total amount of interconnections is minimized.

Authors in [60] presented an approach that tries to control the distribution of the local interconnections. Using partitioning as a post-processing procedure after cut minimization, maximum local interconnection is also minimized at the expense of overall cut quality. A multi-objective cost function, considering overall cut quality and maximum subdomain degree (i.e., maximum block pin count), is used. However, the block size balance constraint is still kept stiff and only the peak interconnection is spread out. Due to the less flexible cell moves during the partitioning process that does not take 'intelligent' block size balancing into consideration, the individual interconnections are not adaptively adjusted according to the block sizes; hence, an optimal (or even) distribution of the interconnections cannot be achieved without an increase of overall cut quality.

4.2 Resource Utilization Model

In this chapter, we use the following resource utilization model, on which we explore the new partitioning solution space where the individual interconnections are adaptively adjusted according to the amount of the logic occupation. For a K-way partition, it is assumed that the equal amount of resources are assigned to K blocks, and the resource utilization of a block is determined either by the total weight of

the cells within the block (block size), or by total weight of the external connections (block pin count). A similar idea is used in the area of white space allocation for congestion removal. The white space allocation approach tries to improve routability of the placement by adaptively injecting the unoccupied white space to potentially congested areas that have relatively more estimated interconnections[64]. In this approach, resource over-utilizations are avoided by padding unoccupied space at the expense of design size increase; and, the size of white space injected corresponds to the amount of routing resource increase.

We introduce a simple yet effective concept of *crowdedness* to give the partitioning the control over the resource utilization distribution. Crowdedness of a block is defined as the weighted sum of the actual block size (i.e., logic occupation area) and the external degree of the block (i.e., the number of nets incident on the block). Crowdedness can be viewed as a virtual complexity metric such that the amount of interconnections of a block accounts for the additional block size, and it directly conforms to our resource utilization model. The crowdedness balanced partitioning gives the flexibility to the partitioning solutions, so that smaller blocks taking relatively smaller amounts of resources are allowed to have denser interconnection, and the blocks with sparser interconnections are allowed to have more logic occupancy. Although our resource utilization model may not be applicable to every case, it still provides an effective means to exploring the undiscovered partitioning solution space using the adaptive interconnection distribution. Figure 4.1 shows the examples where the concept of crowdedness is depicted.

Figure 4.2 shows the typical trend of the distribution of the maximum crowdedness from a number of 8-way partitioning solutions using a state-of-the-art parti-

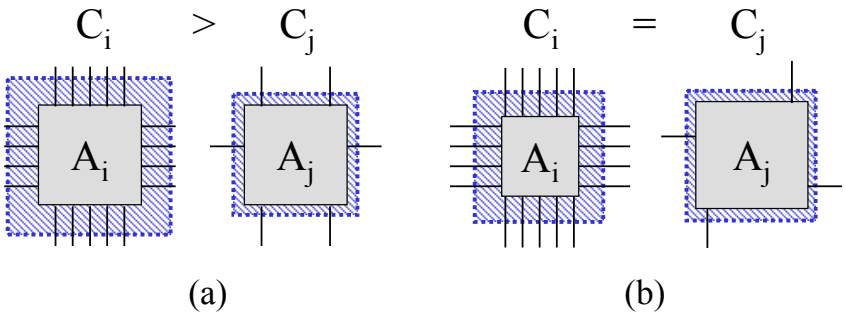


Figure 4.1: Crowdedness comparison of two partitioned blocks, A_i and A_j, based on our resource utilization model. (a)A_i is more crowded than A_j (i.e., A_i has greater resource utilization than A_j). (b)A_i and A_j are equally crowded (i.e., they have the same crowdedness).

tioner, hMetis[39, 41]. It is clearly shown that the conventional partitioner produces partitioning solutions whose maximum crowdedness are far from the optimal. This is interpreted as the indication that some partitioned blocks have larger sizes, and have more pin counts as well, yielding significantly unbalanced resource utilization. The partitioning solutions tend to have better overall cut quality (SOED) as the feasibility conditions imposed by balance constraints are loosened. However, this does not imply that the solutions with looser balance constraints have smaller maximum crowdedness; On the contrary, many solutions with looser balanced constraints are more severely crowdedness-unbalanced, implying potential resource over-utilizations.

In our approach, we simultaneously consider the original objective of the partitioning problem, overall cut quality, and the distribution of crowdedness over the partitioned blocks. If a block turns out to be crowded, we can alleviate the crowdedness either by relocating some logic inside the block to less crowded blocks (i.e., providing more space to the crowded block) or by decreasing the local intercon-

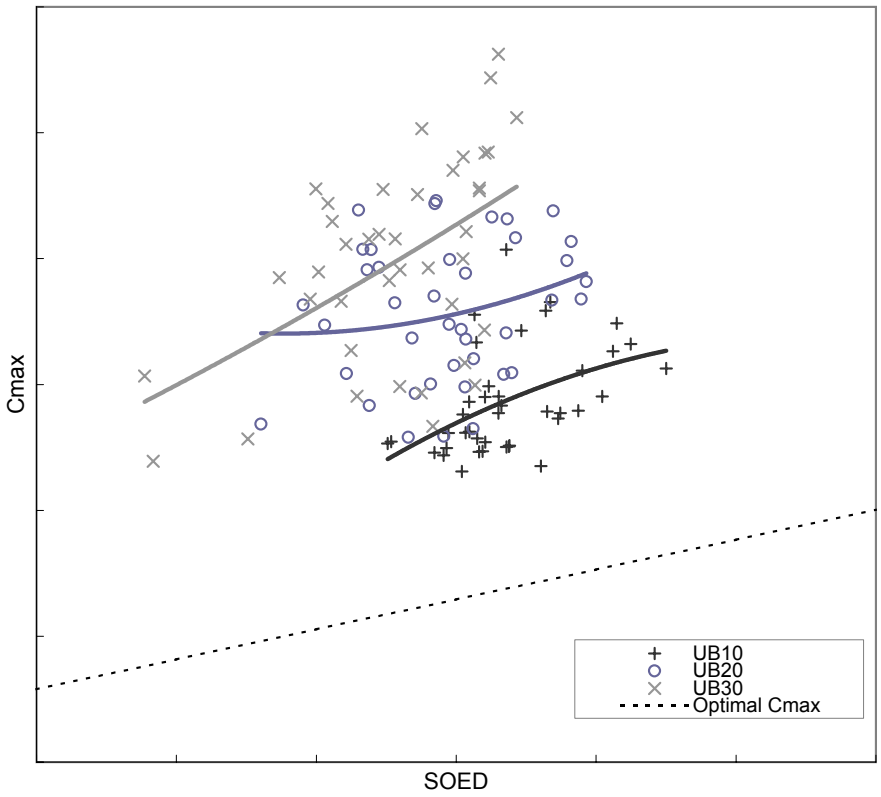


Figure 4.2: Maximum crowdedness C_{max} from 8-way partitioning of ibm05 using hMetis with SOED minimization. UB10, UB20, and UB30 represent block size balance constraints allowing 10%, 20%, and 30% deviations from the average block size, respectively.

nections from the block. In this context, the original block size balancing constraint is intelligently relaxed for crowdedness control. Two partitioning solutions with the identical cut qualities (SOED's) but different crowdedness distributions are shown in Figure 4.3. From the perspective of resource utilization distribution based on our model, it is clear that the partition in Figure 4.3(b) is a more desirable solution than that in Figure 4.3(a). Note that while the maximum crowdedness has been reduced, the maximum external degree remains the same. This example shows that the mini-

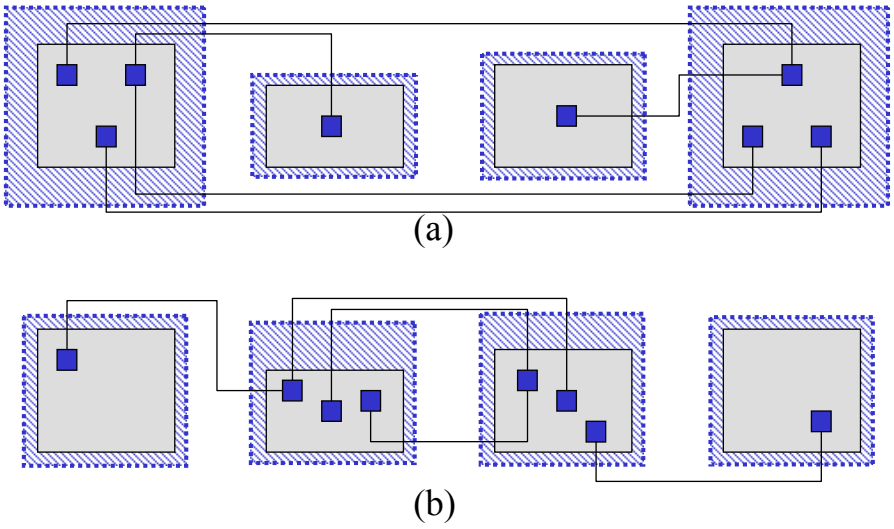


Figure 4.3: The partitioning solutions with the same SOED and the max ED (Only the cells connected to the exposed nets over the blocks are shown) but with different resource utilization distribution. Relocating several cells in the partition (a), the new partition (b) is more crowdedness-balanced.

mization of the maximum external degree is not the solution for resource utilization control, where the block sizes and the external degrees must be considered together. Neither SOED minimization or maximum ED (external degree) minimization is capable of telling the difference between these two partitioning solutions.

The multi-FPGA systems is a typical application of the proposed approach. Thanks to the advance of technology, the hardware-specific pin limit for each FPGA is not a hard constraint any more in multi-FPGA systems. However, an excessive amount of connections required for interacting with other FPGA's still is the critical factor to increase a degree of pin-multiplexing. As a result, greater external degrees lead to more additional logic inside a FPGA to accommodate the multiplexing; consequently, the chances of failure fitting the design into the FPGA's, due to the resource over-utilization, increase. Moreover, the timing performance of the

implemented design on the FPGA's becomes degraded if too many multiplexings are involved. Also, for the general layout in an incremental context, the proposed approach can be viewed as a congestion-aware partitioning, assuming the model that the routing resources are evenly assigned to the equal-sized bins and the size of the unoccupied area (e.g., white space) of the bin corresponds to available routing resources.

In the following sections, we present a new resource utilization balanced partitioning using crowdedness, on top of the multilevel paradigm. The multilevel partitioning has been shown to be the most effective approach to producing excellent partitioning quality in a dramatically shorter run time compared to conventional flat-level partitioning heuristics. Generally a multilevel partitioning consists of 1) multilevel clustering (coarsening), 2) initial partitioning at the coarsest level, and 3) multilevel FM refinement with unclustering (uncoarsening)[39, 41]. During the coarsening phase, the problem size is gradually reduced over the levels while capturing strong connectivity in the circuit netlist. Then, the initial partition at the coarsest level is propagated to lower levels, at which FM partitioning is performed to improve the current initial partition inherited from the upper level.

We first define crowdedness as a partitioning quality metric. Then we describe our multilevel partitioning approach using the crowdedness metric, where the crowdedness distribution is optimized by minimizing the maximum crowdedness. Since we still need to minimize the overall interconnection as well, the cell move policy in the underlying FM partitioning must be carefully designed to account for both objectives. Instead of a parameterized dual-objective cost function, the proposed approach uses a prioritized cell selection, which is simple but helps

avoid being trapped in local minima. Experimental results show that the proposed crowdedness based partitioning achieves near-optimal solutions in terms of crowdedness distribution, while the overall interconnection quality also is improved, but the original feasibility condition is barely violated. Such solution space never has been discovered with the conventional partitioning approaches.

4.3 Crowdedness

In this section, the definition of crowdedness is introduced as a new quality metric of a partitioning solution.

Definition 4.1. A circuit is modeled by a netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$, where $\mathcal{C}$ is a set of *cells* with associated sizes and $\mathcal{N}$ is a set of *nets* (hyperedges) connecting two or more cells. A *pin* is a connection point between a cell and a net.

Definition 4.2. A *K-way partition* of a hypergraph is described by the K-tuple, $P = (A_1, A_2, \cdots, A_K)$ where $A_i \cap A_j = \emptyset$ for $i \neq j$ and $\cup_i A_i = \mathcal{C}$. A_i is said to be the i-th block of the partition.

Definition 4.3. For a partition $P = (A_1, A_2, \cdots, A_K)$ on hypergraph H, *crowdedness* of a block A_i is a virtual complexity metric where the block size and the pin count are combined, and is defined as

$$C(A_i) = S(A_i) + \alpha_i E(A_i)$$

where $S(A_i)$ is the physical size occupied by the cells belong to A_i, $E(A_i)$ is the external degree of A_i (i.e., the number of nets incident on A_i connecting to other block(s)), and α_i is a positive weighting coefficient. $E(A_i)$ is also referred to as block

pin count of A_i. For simplicity, we use C_i, S_i, and E_i instead of $C(A_i)$, $S(A_i)$, and $E(A_i)$, respectively, when the index of the block is clear from the context.

Crowdedness of a block can be viewed as the original block size added by a virtual block size that is proportional to the pin count (the external degree) of the block. The positive constant α_i determines relative sensitivity on the amount of the external degree for block A_i. For example, if α_i is the same for every block A_i, the cost of using each block pin is identical all over the blocks. However, assigning a relatively higher value of α for a certain block A_i, the resource utilization is customized such that, for the block A_i, the crowdedness is reduced more by E_i reduction rather than by S_i reduction, since the use of any pins of A_i is more expensive than the pins on the other blocks. Note that the minimization of overall crowdedness (minimize $\sum_i C_i$) is an equivalent objective to the SOED minimization (minimize $\sum_i E_i$) because $\sum_i S_i$ is constant.

Definition 4.4. Consider a partition $P = (A_1, A_2, \cdots, A_K)$ on hypergraph H, with SOED E_{tot} $(= \sum_i E_i)$ and the maximum crowdedness C^* $(= \max_i C_i)$. P is said to be perfectly crowdedness balanced at E_{tot}, if there exist no other partitions with the same SOED E_{tot} have maximum crowdedness values smaller than C^*. Given a SOED E_{tot}, the *optimal* (minimal) maximum crowdedness C^* is achieved when crowdedness is evenly distributed over blocks. Hence, we have

$$C^* = \sum C_i/K = \sum S_i/K + \alpha \cdot E_{tot}/K$$

where the optimal maximum crowdedness is a linear function of SOED.

4.4 Multilevel Partitioning Using Crowdedness

In this section, we define a new partitioning problems that consider crowdedness distribution as well as the original overall cut quality. The proposed approach, for the improvement of crowdedness distribution, can be incrementally applied to a previous partitioning solution, where only overall interconnection has already been minimized. Otherwise, the approaches can be utilized in stand-alone partitioners starting from scratch, that optimize cut quality and crowdedness distribution simultaneously.

A simple idea to take crowdedness balance into consideration using crowdedness is keeping the original partitioning objective, SOED, but applying a crowdedness balance constraint instead of a block size constraint. With the SOED objective and the crowdedness constraint, we formulate a partitioning problem as follows.

Problem 4.1. Given a netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$, the K-way partitioning problem consists of finding a partition of K blocks ($K \geq 2$) such that the $\sum_{i=1}^{K} E_i$ is minimized while each C_i is constrained to a certain size, i.e., $C_{MIN} \leq C_i \leq C_{MAX}$.

Even though the problem is intuitively clear, there are some difficulties in this approach. First, the decision on the reasonable values of C_{MAX} for feasible solution is not easy since it depends on the characteristics of a netlist hypergraph such as density of the hypergraph. Second, given a crowdedness constraint, the feasibility of the resulting partition is not guaranteed, i.e., the final solution is not always bounded by the specified crowdedness. For example, we may want to start the iteration of cell moves from a crowdedness-balanced partition to results in a final feasible solution. But for overly tight crowdedness constraints, even construction of

Algorithm `ML_crowdedness_constraint`

Input: netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$,
(optional) previous partition P_{prev} on H
Output: K-way partition P

1. Construct a t-level clustering tree from H
2. Obtain a top level initial partition P_t (SOED minimized)
3. Set C_{MAX} to $\max_i C_i$ in P_t
4. **for** $l = t - 1$ down to 0 **do**
5. **if** C_{MAX} is not violated in P_{l+1}
then $C_{MAX} = (1 - x)C_{MAX}$
6. P_l = `K-way_FM`(H_{l+1}, P_{l+1}, C_{MAX})

Figure 4.4: Outline of multilevel partitioning with crowdedness constraint.

such an initial partition is not trivial.

Recall that a partition which is more crowdedness balanced is preferred, and our approach is implemented on the multilevel partitioning paradigm. Therefore, it can be suggested that the use of dynamic crowdedness constraints over the partitioning levels as shown in Figure 4.4. First, at the coarsest level, overall cut quality is optimized by SOED minimization by K-way FM partitioning [33]. Then, the peak crowdedness of the top level partitioning solution is used as the crowdedness constraint, and the constraint is gradually reduced while proceeding down to lower levels (x in Figure 4.4 was set to 0.02 for our experiments). If the crowdedness constraint is violated in the current level partition, the constraint is not modified but tried again at the next lower level. In the Figure 4.4, the crowdedness is not actively rearranged, since it does not have an actual cost functions to drive the crowdedness balancing. We use the result from this approach as a less-effort solution for comparison purpose in the following section.

Unlike the above preliminary approach, the overall crowdedness (equivalently, SOED) and the peak crowdedness can be simultaneously minimized using

Algorithm `ML_crowdedness_cost`

Input: netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$,
(optional) previous partition P_{prev} on H
Output: K-way partition P

1. Construct a t-level clustering tree from H
2. Obtain an initial partition P_t (SOED minimized)
3. P_t = `K-way_FM`(H_t, P_t) with the crowdedness cost
4. **for** $l = t - 1$ down to 0 **do**
5. P_l = `K-way_FM`(H_{l+1}, P_{l+1}) with the crowdedness cost

Figure 4.5: Outline of multilevel partitioning with crowdedness cost

two objective functions for each of them. They can be combined to form a single multi-objective cost in a parametric way, or can be separated but carefully managed in a prioritized method. With the problem formulation below, we do not have the feasibility (in terms of crowdedness balance) issues since now crowdedness is a part of the objective function.

Problem 4.2. Given a netlist hypergraph $H = (\mathcal{C}, \mathcal{N})$, the K-way partitioning problem consists of finding a partition of K blocks ($K \geq 2$) such that both $\sum_{i=1}^{K} E_i$ and $\max_{1 \leq i \leq K} C_i$ are minimized.

One way of solving this problem is to set up and minimize a multi-objective cost function combining these qualities, $\sum_{i=1}^{K} E_i + R(\max_{1 \leq i \leq K} C_i)$, where R is a positive weighting factor that determines relative emphasis of maximum crowdedness. However, $\sum_i E_i$ cannot be totally isolated from $(\max_i C_i)$ and they are actually closely related, so the parameter R must be carefully chosen.

As an alternative technique which is non-parametric, we suggest a priority-based approach in which maximum crowdedness is the highest priority objective, and SOED (i.e. overall crowdedness) is the second. We always look for a new

solution from a current solution that has a smaller value of maximum crowdedness. However, moves to a solution with a smaller values of SOED than the current SOED are also allowed if maximum crowdedness cannot be improved. More specifically, in the k-way FM refinement at each level in the multilevel partitioning, a vertex (i.e., cluster) is randomly chosen, and the move of the vertex to a target block is always taken if the move of the vertex yields a maximal reduction of the peak crowdedness and the reduction is positive. If, however, any moves of the vertex cannot reduce the maximum crowdedness, the move with a maximal positive SOED reduction is taken even though it will increase the peak crowdedness. If a current vertex cannot make any moves that reduce either costs, we look for another candidate vertex to move. According to our extensive experiments, this prioritized approach dramatically helps avoid getting trapped in local minima. Also, this is more suitable to incremental contexts where the crowdedness control is applied to a previous partitioning solution in which SOED has been well-minimized [25, 19].

4.5 Experimental Results

We implemented our algorithms in C++/STL and evaluated the performance on 18 ISPD benchmark circuits [1]. For each circuit, the performance of the proposed algorithms is evaluated by incrementally applying the crowdedness-based partitionings to the best solution from 20 runs of K-way hMetis with the SOED minimization objective.[1] The proposed algorithm re-partitions each of the netlist hypergraphs with the capability of redistributing resource utilization (represented by the crowd-

[1]The best of 20 runs of partitioning was obtained not because it is practically useful, but because the tight initial solution renders the problem more difficult to optimize both SOED and $\max C_i$.

edness). For each test case in our experiments, the weighting coefficient α_i was set to $\sum S_i / \sum E_i$ for every block A_i from the partitioning solution from hMetis, which makes the contribution of a block pin count to the crowdedness value equivalent to the size of the block. For instance, if a block in a new partition has the average block pin count of the previous SOED-minimized partition, then the block pin count will be reflected as the average block size, in the virtual complexity of crowdedness. The maximum crowdedness and SOED, before and after applying our algorithms, are compared in Table 4.1.

Note that only half of the change of an external degree is reflected in the crowdedness, and there is a lower bound of the maximum crowdedness at a given SOED. To fairly evaluate the effectiveness of our approach, two measures are additionally suggested and shown in the third and the fourth columns for each algorithm — effective SOED and the deviation from the optimal crowdedness. The effective SOED represents the equivalent value of SOED that we *would* need in order to reach the same crowdedness as the resulting partition, *if* the external degrees were uniformly reduced over the blocks. This metric indicates the effort needed to have the same maximum crowdedness without any awareness of crowdedness distribution. Therefore, the difference from an effective SOED to an actual SOED quantifies the effectiveness of the proposed algorithms. The fourth column for each algorithm shows the resulting partitions' deviations from perfectly crowdedness-balanced solutions at their current SOED, which indicates how close the resulting partitioning solutions are to the optimal solutions.

As shown in Table 4.1, both the maximum crowdedness and SOED are improved by our crowdedness-based partitioners. Whereas the overall interconnection,

K		hMetis		crowdedness constraint				crowdedness cost			
		C_{max}	SOED	C_{max}	SOED	Eff.SOED	$C_{max} - C^*$	C_{max}	SOED	Eff.SOED	$C_{max} - C^*$
4-way	min	1	1	0.901	0.897	0.760	0.058	0.889	0.921	0.732	0.002
	ave	1	1	0.968	0.952	0.920	0.145	0.918	0.965	0.806	0.069
8-way	min	1	1	0.951	0.896	0.880	0.083	0.818	0.921	0.533	0.000
	ave	1	1	0.970	0.929	0.914	0.263	0.895	0.984	0.716	0.170
16-way	min	1	1	0.931	0.864	0.797	0.080	0.782	0.912	0.416	0.021
	ave	1	1	0.966	0.928	0.907	0.238	0.875	0.986	0.674	0.134
32-way	min	1	1	0.959	0.884	0.871	0.140	0.753	0.977	0.247	0.099
	ave	1	1	0.978	0.941	0.935	0.349	0.862	1.001	0.588	0.174
64-way	min	1	1	0.963	0.860	0.882	0.174	0.721	0.946	0.554	0.095
	ave	1	1	0.974	0.923	0.950	0.393	0.877	0.981	0.793	0.224

Table 4.1: Comparison of relative maximum crowdedness and sum of external degrees (SOED) to the best partitioning results from 20 runs of hMetis with SOED minimization.

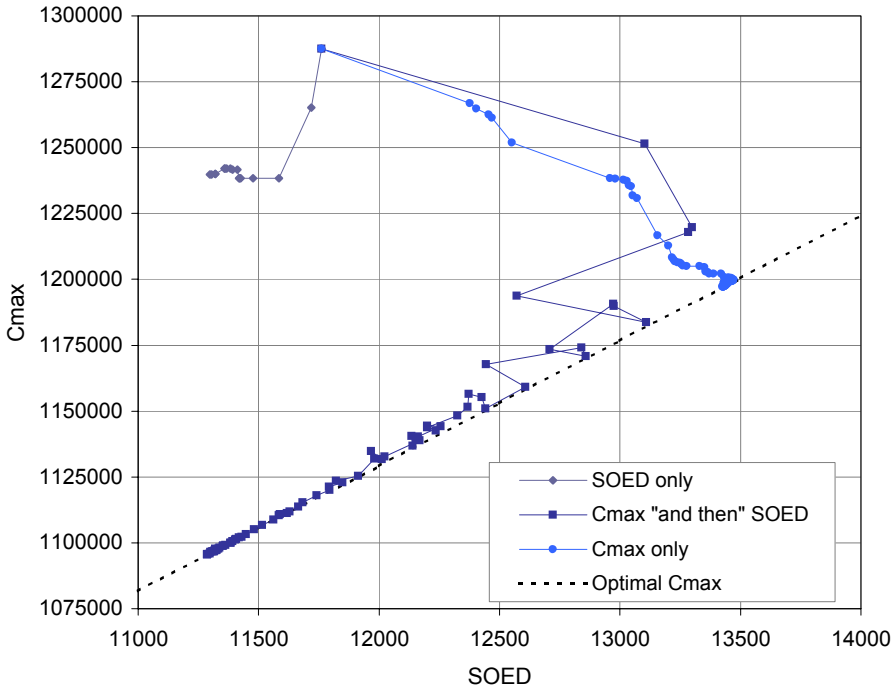


Figure 4.6: Solution space traversals from the multilevel partitionings with different objectives.

represented by SOED, is more reduced by the crowdedness constraint-based approach, the partitioning with crowdedness cost using the prioritized cell move policy shows a larger reduction of maximum crowdedness than the partitioning with dynamic crowdedness constraint. The actual SOED is close to the effective SOED in the results from the crowdedness constraint-based partitioner, since the crowdedness cost does not drive the solutions, but rather the maximum crowdedness reduction is obtained naturally from the overall crowdedness (SOED) optimization at the same rate.

The effectiveness of crowdedness control is improved by using the crowdedness cost. Even if the actual SOED reduction is not so significant, the effective

SOED's show that the rearrangement made by our algorithm has an equivalent impact on the crowdedness distribution that would have been obtained by the large amount of SOED reduction (25%–75%). Also, the resulting solutions are very close to the optimal solutions (i.e., perfectly crowdedness balanced partitions). According to the experimental results, about 40% of the resulting solutions are within 5% from the optimal solutions in terms of maximum crowdedness.

The effectiveness of the proposed algorithm is clearly recognized by demonstrating the solution space traversals shown in Figure 4.6. In this example, the maximum crowdedness and SOED are plotted during the entire move sequence of three different multilevel partitioning algorithms applied on a partitioning solution from hMetis. Applying conventional "SOED only" minimization (which is called V-cycle in hMetis), the crowdedness distribution is barely improved and still is far from the optimal solution. Furthermore, if we try to optimize crowdedness distribution by minimizing the maximum crowdedness without considering SOED quality, the resulting partition has more improved crowdedness, but is trapped in a local minimum that has much larger value of SOED. Thanks to our prioritized cell move policy that accounts for both the maximum crowdedness and SOED, it is able to find a solution that has a comparable reduction of SOED to SOED-only partitioning, while the crowdedness distribution is optimal, i.e., the crowdedness is perfectly balanced. As shown in Figure 4.7, the physical block sizes and the external degrees are adaptively rearranged to produce an optimal solution in terms of crowdedness distribution at a new SOED that is also further improved from the previous SOED. For the experimental purpose, the original feasibility condition, block size balancing, was intentionally removed to give the proposed algorithm the maximal flexibility for

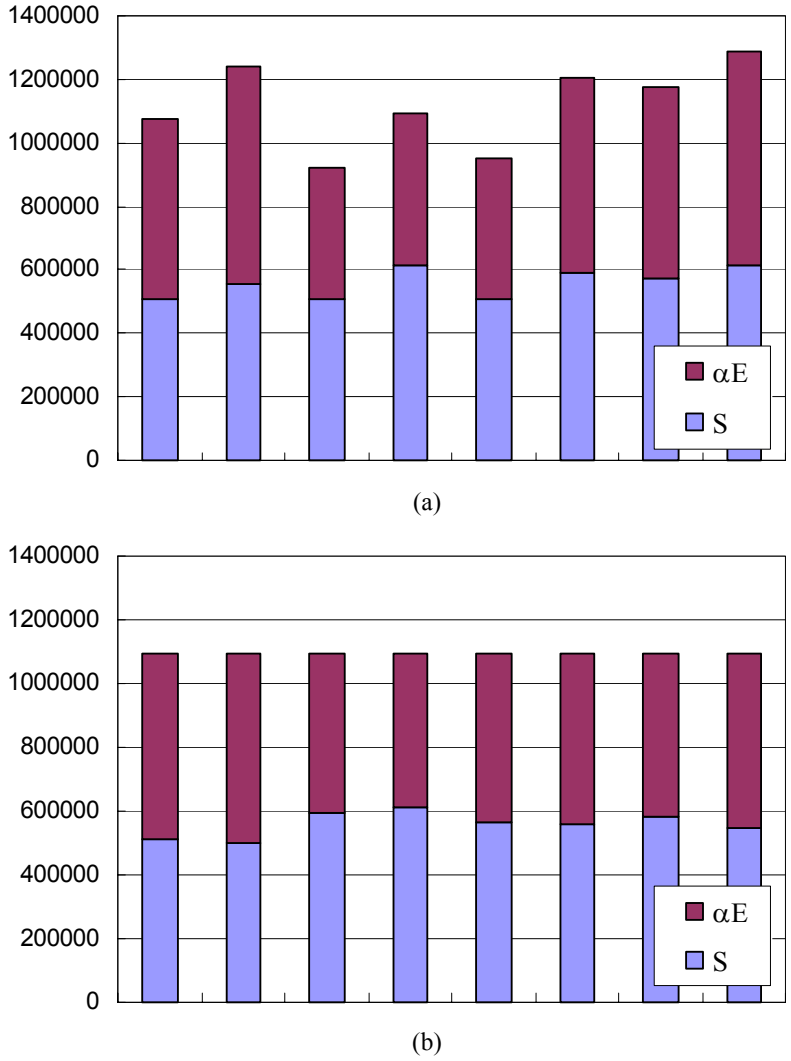


Figure 4.7: Crowdedness distribution (ibm05) over 8 partitioned blocks (a) from hMetis with SOED minimization (SOED = 11762), and (b) after applying the proposed partitioning algorithm (SOED = 11286). The original block size balancing constraint, 10% deviation from the average block size, is not violated.

crowdedness control. However, any severe violations of the block size balancing were not observed; the deviations from the maximum block size in the original partition from hMetis ranged only up to 10%.

Figure 4.8 shows a more detailed diagram on how the proposed algorithm drives the solution, minimizing both the maximum crowdedness and SOED. Note that our prioritized move policy does not allow a cell (or cluster) to move to another block if it degrades both qualities. However, when the maximum crowdedness cannot be improved, a move with SOED reduction is taken as an alternative. At higher levels in the proposed multilevel partitioning where each cluster (vertex) is relatively large compared to the block sizes, it inevitably involves SOED increases since the block size change is more dominant in the crowdedness. By traversing back and forth from the optimal solution, the clusters are well shuffled for crowdedness balancing, and the nets are disentangled for SOED reduction. While the cluster sizes become smaller at lower levels after successive unclustering, both the crowdedness distribution and SOED start to improve, maintaining the near-optimal crowdedness distribution at the corresponding SOED value. Allowing the moves of the clusters with SOED reduction but maximum crowdedness increase, the solution does not fall into local minima and looks for a better solution by giving up an optimal solution at a higher SOED value. The repetition of this sequence is represented as saw-tooth shapes along the optimal solutions in Figure 4.8.

4.6 Conclusion

In this chapter, we presented a K-way multilevel partitioning for uniform resource utilization. Based on our resource utilization model, a new quality metric of a

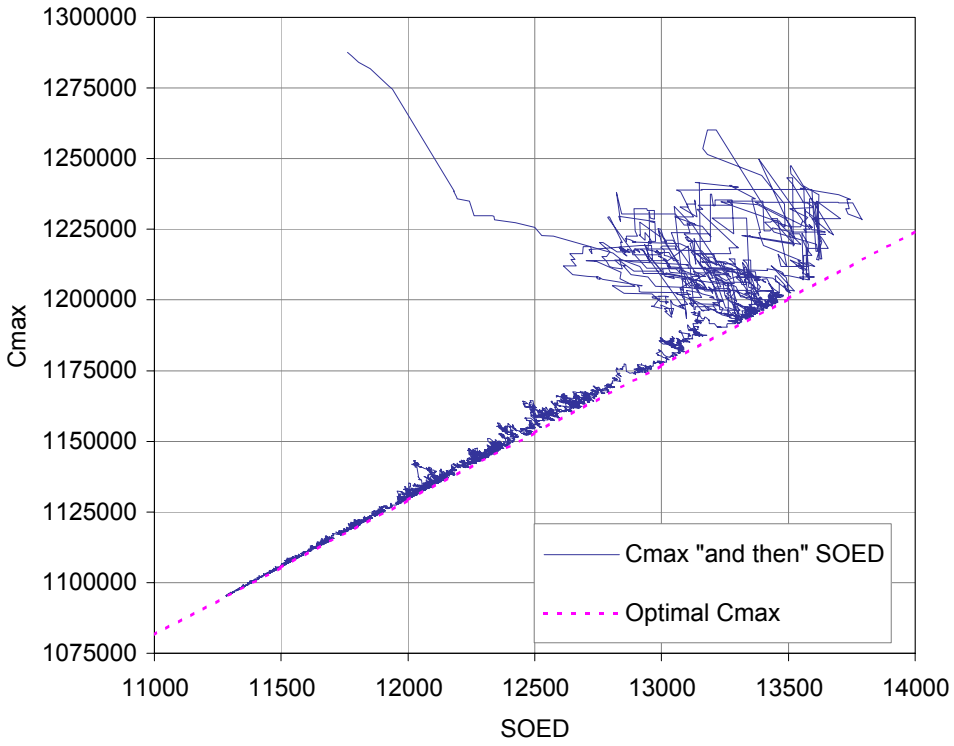


Figure 4.8: Detailed solution space traversal.

partitioning solution, *crowdedness*, was defined as the weighted sum of logic occupation and the local connectivity. Experimental results show that our crowdedness-balanced multilevel partitioning produces near optimal solutions in many test cases in the sense that the crowdedness is balanced over the partitioned blocks, while the overall interconnection is not sacrificed. The proposed algorithm aimed at the uniform distribution of the crowdedness to avoid potential resource over-utilizations. More investigation for user customizable resource utilization will be performed as our future work. One way is to use a resource priority; a user can specify different relative costs of interconnection resources for different blocks by assigning different weighting coefficients. We can also extend it to *crowdedness shaping*, if we want a

particular user-tailored pattern of crowdedness distribution rather than a uniform distribution.

Chapter 5

Multilevel Multirate Partitioning

The multilevel partitioning paradigm has recently been introduced, and received a lot of attention because of its dramatic improvement in both cut quality and run time. Though the multilevel partitioning paradigm is relatively robust, it has limited flexibility; while the partitioning solution is refined over multiple levels, the problem instance at each level totally relies on the prior construction of the multilevel clustering tree. Specifically, even though the information on the local connectivity qualities in the clustering tree is available at the time of uncoarsening, the multilevel partitioning does not restructure the tree to favor better FM refinement. In this chapter, we present a more flexible version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree; the multilevel uncoarsening phase involves *multirate unclustering*, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels.

By the implicit restructuring of the clustering tree during the uncoarsening and refinement phase, the proposed approach produces higher quality solutions and more predictable quality/runtime trade-offs.

5.1 Introduction

As designs become massively interconnect-dominated and present unmanageable instance complexities, circuit partitioning is recognized as a critical optimization problem in computer-aided VLSI design automation. Partitioning solutions have a great impact on automatic placement and routing procedures, especially in large scale design procedures, where tens of millions of cells are handled. Millions of the cells cannot be handled in a flat mode any more due to the limitation of computation power and memory space. Moreover, concurrent engineering of the individual subcircuits can shorten design turnaround time. As a result, a circuit needs to be partitioned into several blocks where the massive amount of data is broken into manageable sizes while minimizing the interactions between the partitioned blocks. Partitioning techniques are often embedded in the large scale placement procedure to determine the optimal global positions [12, 9, 38].

The multilevel partitioning paradigm has recently been introduced, and received a lot of attention because of its dramatic improvement in both cut quality and run time [2, 39, 41, 20]. In particular, the speed-up obtained from the multilevel partitioning enables commercial CAD tools to achieve high quality partitioning solutions in a practically tractable time. The multilevel partitioning consists of three phases: multilevel coarsening, initial partitioning at the coarsest level, and multilevel uncoarsening with FM refinement During the coarsening phase, the problem

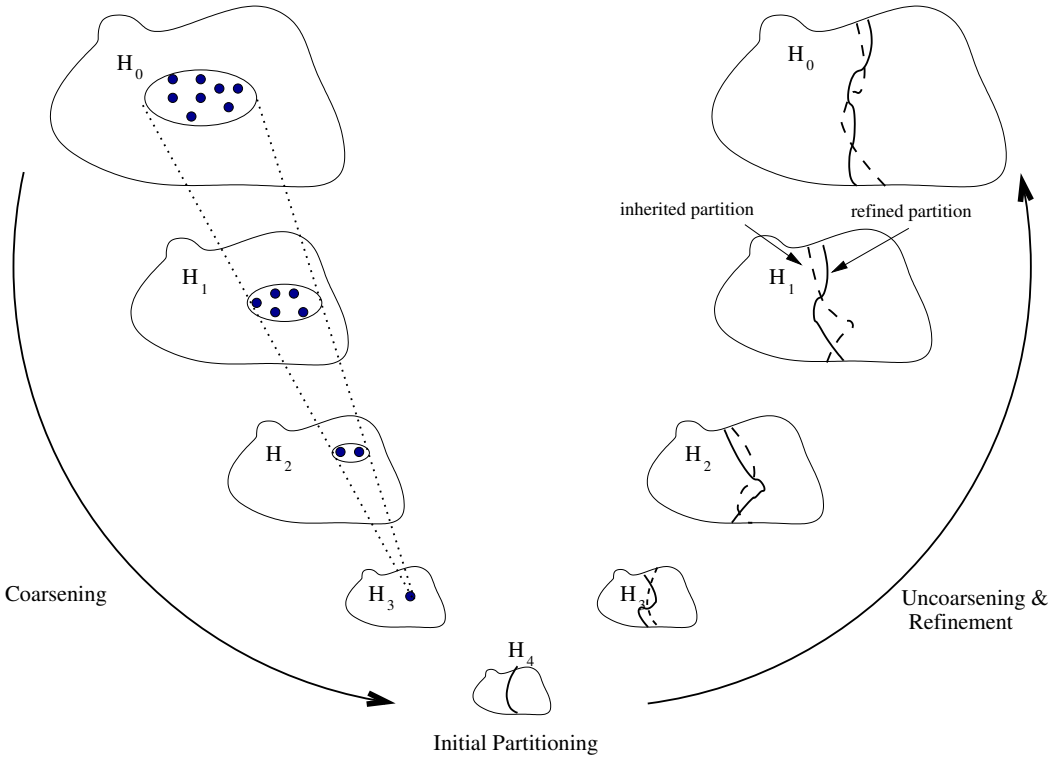


Figure 5.1: Multilevel hypergraph bi-partitioning. H_i is the next level coarser hypergraph of H_{i-1}.

size is gradually reduced over the levels while capturing strong connectivities in the circuit netlist. At the coarsest level, a relatively high quality initial partitioning solution is quickly obtained. Then, the current partitioning solution is successively propagated to lower levels, where the partitioning solution on the bigger problem keeps improving by an iterative improvement based refinement heuristic.

Though the multilevel partitioning paradigm is relatively robust, we notice that it has limited flexibility; while the partitioning solution is refined over multiple levels, the problem instance at each level totally relies on the prior construction of the multilevel clustering tree. As shown in Figure 5.1, a vertex (cluster) in a coarser

hypergraph at a higher level, actually is a set of several bottommost leaf cells. The coarsening is performed in a hierarchical manner constructing a multilevel clustering tree, so that the local connectivities of the lower level clusters (or leaf cells) are not seen until the level-by-level unclustering procedure reaches the level at which those clusters are located. Even though the information on the local connectivity qualities in the clustering tree is available at the time of uncoarsening, the multilevel partitioning does not restructure the tree to favor better FM refinement.

It is noticed that further improvement of the multilevel partitioning based on the trade-off between run time and quality have barely been introduced. Adjusting the problem size reduction ratio between the adjacent levels, the number of levels can increase so that possibly better quality solutions can be achieved by more refinement steps on more levels at the cost of longer run time [2]. However, through the extensive experiments, it is observed that the increase of the number of levels does not provide a reasonable trade-off, and the higher number of levels does not guarantee a higher quality solution.

This limitation of the flexibility is an inherent obstacle against fundamental improvement of the multilevel partitioning. In this chapter, we present a new version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree constructed during the coarsening phase; the multilevel uncoarsening phase involves *multirate unclustering*, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels. By the implicit restructuring of the clustering tree during the uncoarsening and refinement phase, the proposed approach produces higher quality solutions and more predictable quality/runtime trade-offs.

5.2 Multilevel Multirate Partitioning

In this section, we analyze the multilevel partitioning paradigm and suggest a simple yet effective approach to providing flexibility to the multilevel partitioning based on multirate unclustering.

5.2.1 Anatomy of Multilevel Partitioning

All the iterative improvement based partitioning (IIP) heuristics are based on the greedy strategy; they start with some feasible (balanced) solution and iteratively make a move to the best possible neighboring solution at a time. The process is performed until the algorithm reaches a local minimum, i.e., a solution for which all neighboring solutions have greater costs. However, the IIP heuristics – such as Fiduccia-Mattheyses (FM) algorithms and their variations – have some mechanisms to avoid being trapped into local minima. In an entire sequence of moves of all the cells, every move is made by the best cell with a highest cost reduction (gain) at each moment, whether the gain is negative or positive. After the completion of one pass, the best partition with a minimum cost ever is backtracked and selected as the initial partition for the next pass (See Figure 5.2). This process is performed until no more cost reduction is available during a pass.

Even though it allows hill-climbing out of local minima, IIP approach has two basic limitations; as the problem sizes become greater, the number of passes tend to increase, albeit the cost reductions in most passes are not high enough. Furthermore, the resulting quality of the final partition is greatly dependent of the very first initial partition, yielding a wide variance of the qualities when run multiple times with different initial partitions. The multilevel partitioning effectively addresses both

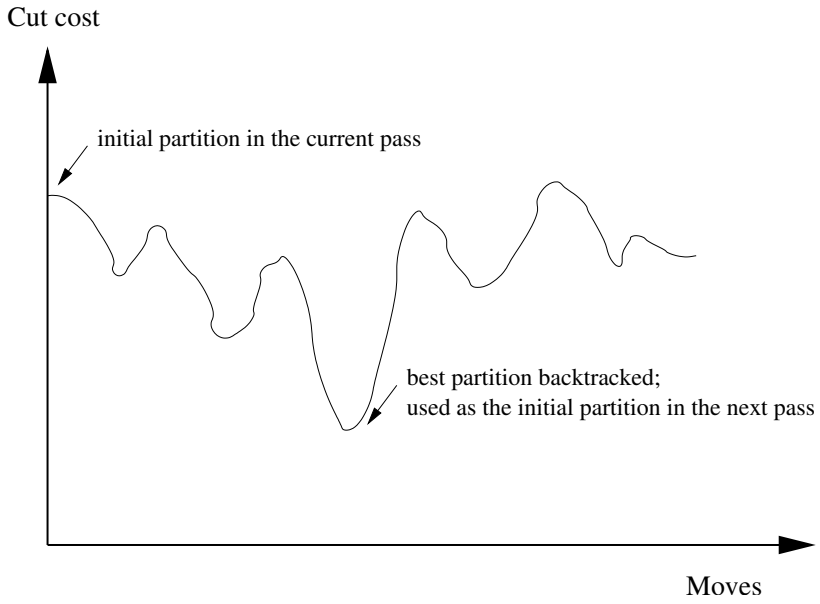


Figure 5.2: Cost variations by the cell moves in a pass in FM algorithm.

issues even though its underlying refinement algorithm is still FM partitioning.

The original problem size is gradually reduced level by level, to the extent that FM algorithm can be efficiently applied. At each level, connectivity-driven clustering tries to minimize the number of exposed nets (hyperedges) at the current level, during the construction of multilevel clustering tree. When the problem size has been reduced enough to satisfy a given specified condition, or no more significant problem size reduction is available, the multilevel clustering tree construction is completed. While the clusters are uncoarsened traversing down to the lower levels from the top, the computation effort required for the refinement is associated with the quality of the current initial partition from the upper level and the problem size of the current level. A high quality initial partition is easily obtained at the top level, thanks to the connectivity driven clustering process. Since the initial partition keeps improving while passing through the intermediate levels, the increasing problem size

tends to be counterbalanced with the decreasing cost of the initial partition, so that the computation effort is not significantly increased at the lower levels.

Our attention focuses on the following aspects of the multilevel partitioning. First, note that a single move of one cluster at a higher level corresponds to a group migration of the many leaf cells. This process acts as if it traverses to the states in the original solution space in a wide stride, where it would have been difficult to be searched without considerable amount of successive moving of those leaf cells at the bottom level. For instance, consider a move of a cluster with n leaf cells to another block. Then, the resulting state transition made by a single move of the cluster would have been done by a series of n moves at the bottom level. There are $n!$ different ways to achieve the same state transition. On the other hand, due to this implicit successive group migrations, the hierarchical structure of the clustering tree tends to bias the partitioning solution throughout the multilevel refinement steps. As the FM refinement proceeds down to the lower levels, it becomes more difficult to climb out of the current local minimum since most cells settle down in the current positions which were inherited from upper levels. For example, there may exist a lower-level cluster which is desirable to be split for overall quality, but remains hidden in the enclosing parent clusters during the refinement phase until very fine levels are reached.

5.2.2 Multilevel Partitioning with Multirate Uncoarsening

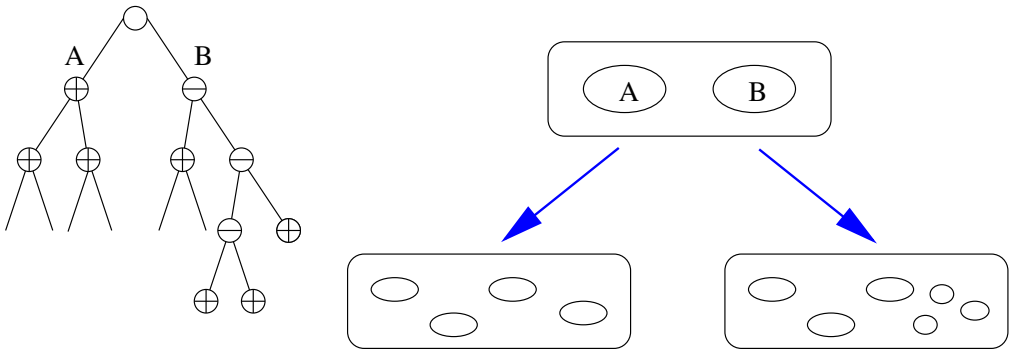
The proposed approach starts with the same multilevel clustering method as that of the state-of-the-art multilevel partitioner, hMetis [39]. We use a connectivity cost and merging policy similar to FC coarsening in hMetis [41, 42]. Once the

clustering tree construction is completed, Rent exponent [54, 52, 36] is used as a quality indicator of each cluster's local connectivity.[1] In [20], the authors used Rent exponent (also known as Rent parameter) to quantify the physical connectivities of the hierarchical elements in a design hierarchy, and used the hierarchical groupings with relatively strong connectivities as the clustering boundaries in the multilevel coarsening.

Note that, a small Rent exponent for a cluster implies relatively high connectivity inside the cluster and a large Rent exponent implies low internal connectivity but more connectivity to outside of the cluster. In the proposed approach, the quality of every cluster is examined by applying Rent exponent computation, at each level. And, the worst $x\%$ among the current level clusters, that have relatively sparser internal connectivities than other peer clusters but stronger connectivity to outside, are marked as ill-formed clusters. Such a cluster is called a negative cluster, otherwise, it is called a positive cluster. In our experiments, the clusters within the worst 5% are selected at every level.

Once the clustering tree mark-up is finished, an initial partition is obtained at the top level by choosing the best solutions from multiple runs of FM, which is followed by successive multilevel unclustering and refinement steps. However, unlike the level-by-level unclustering in the conventional multilevel partitioning, the unclustering is performed at various rates for different clusters. First, all clusters are unclustered to their immediate children (immediate subclusters) located at the next lower level, which is identical to one-level unclustering. Then, the negative clusters among the new subclusters are further unclustered as follows: Consider

[1] For details on Rent's rule and Rent exponent, which have been widely used for interconnection complexity estimation, refer to [54, 52, 36].



(a) part of clutering tree (b)one−level unclustering (c) multirate unclustering

Figure 5.3: Unclustering of two cluster A and B in (a) clustering tree. (b) Simple one-level unclustering and (c) multirate unclustering, where negative clusters are further unclustered until there are no more negative clusters.

that a (sub)cluster C is negative (i.e., turned out to be an ill-formed cluster). The cluster C is recursively flattened down to the lower levels, until all the positive descendants of the cluster are discovered in the subtree rooted at C. The such positive descendants of the cluster C is now located in place of C. (As an alternative option, we can simply assign the ill-formed clusters some integer values greater than one as the unclustering depth, i.e., unclustering rate.) Upon completing these processes for all the negative clusters at the current level, the problem size relatively increases and the modified hypergraph netlist only consists of positive clusters and the associated nets. Note that, due to the higher-rate unclustering of the ill-formed clusters, some nets that were originally hidden at the current level are now exposed earlier.

FM refinement is then applied to the modified netlist hypergraph, and the resulting partition potentially can be favored with finer granularities of some (originally) lower-level clusters and nets. The nets connecting those cells, which were

Procedure `iml`

Input: a given netlist hypergraph hypergraph $H = (\mathcal{C}, \mathcal{N})$
Output: bipartition of C, $P_0 = (\mathcal{A}_1, \mathcal{A}_2)$

1. Construct a k-level clustering tree T from $H_0 = H$
   // H_i is the resulting hypergraph at level i
2. Obtain the initial partition P_k by multiply applying FM on H_k
3. $H'_k := H_k$
4. **for** $i = k - 1$ down to 0 **do**
5. $S :=$ { immediate children of the clusters in H'_{i+1} in T }
5. $H'_i := \emptyset$
6. `uncluster_further`(S, H'_i)
7. Project P_{i+1} onto H'_i
8. $P_i :=$ `FMbipartition`(H'_i, P_{i+1})

Procedure `uncluster_further`(V, V')

1. $P :=$ positive clusters and leaf cells in V
2. $N := V - P$
3. $V' := V' \cup P$
4. **if** $N \neq \emptyset$
5. $S :=$ { immediate children of the clusters in N in T }
6. `uncluster_further`(S, V')
7. **else**
8. return

Figure 5.4: Multilevel partitioning using multirate unclustering.

not easy to effectively contract during the multilevel clustering phase, are exposed earlier at a higher level, and we have more chances to optimize those connections in the successive lower levels. The computation effort for those problematic nets is transfered from the clustering process to the FM partitioning refinement process; these nets can be viewed as requiring more attention and hence having more priorities since their optimization is tried on more levels than other nets. Although the original multilevel paradigm is still kept, some ill-formed clusters are unclustered at the faster rates than the others; hence, they have more chances to be refined by FM at the cost of more computational effort of the FM heuristic. The above procedure causes the original clustering tree to be implicitly restructured throughout

the proposed multilevel uncoarsening phase. The entire procedure above is called multilevel multirate partitioning (See Figure 5.4).

5.3 Experimental Results

We implemented our algorithm in C++/STL and evaluated the performance on nine ISPD benchmark circuits [1]. For each circuit, we compare the cut quality and run time to those of the regular multilevel partitioning implementation (our own implementation of hMetis [39, 41]).[2] For fair comparison, it should be assured that both implementations construct the same clustering tree. Hence, an identical random seed was provided to a pair of both runs, to guarantee that entire multilevel clustering processes are identical.

Figure 5.5 shows the relative cut quality of the multilevel multirate bipartitioning compared to the multilevel bi-partitioning without multirate unclustering. Since our experimental environments forced both partitioners to use the same clustering tree for uncoarsening and refinement phase, the quality variances are primarily caused by the effectiveness of the proposed multirate unclustering. Thanks to the multirate uncoarsening, the cut qualities have been consistently improved by a range of 2 to 5% cutsize reduction, at the expense of 15 to 28% longer run time. The increased run time corresponds to the problem size increase at each level by a number of lower level clusters and nets which are exposed earlier. This shows that more computational efforts have been made by FM partitioning with the multirate

[2]We cannot compare our results directly to the results from hMetis, since we are unable to create the identical clustering tree in hMetis. Throughout the extensive experiments, our own implementation of hMetis has shown similar or slightly better performance than the original hMetis, in cut quality. However, it takes around 20 to 40% more time because of difference of implementation details.

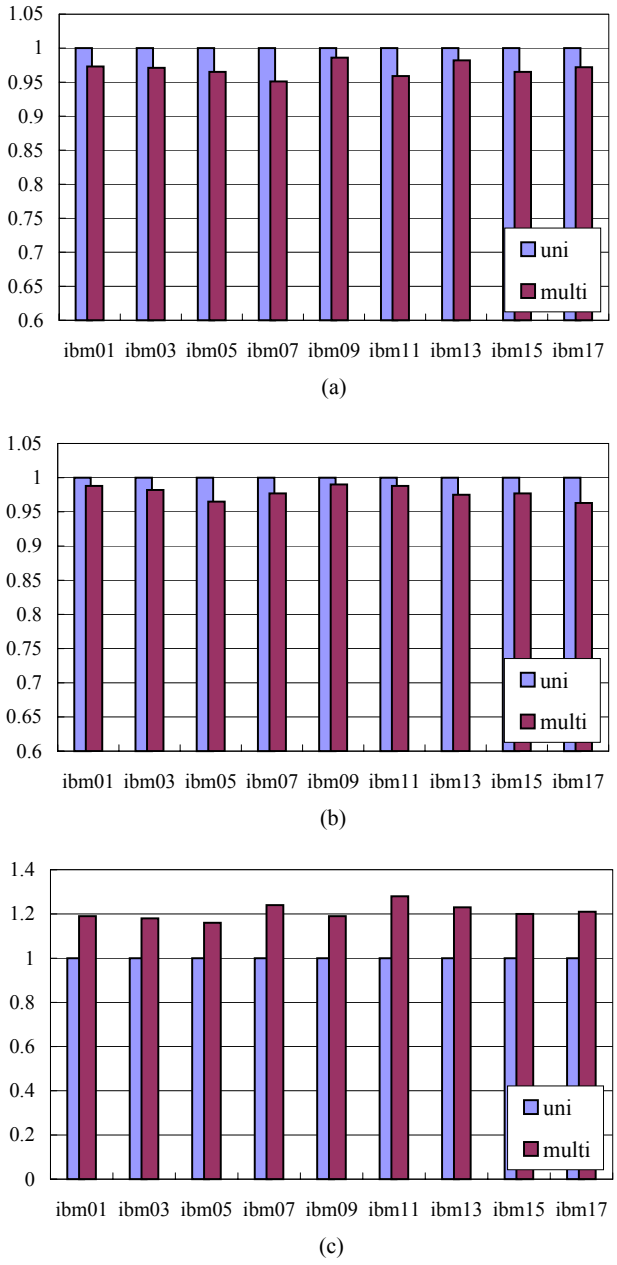


Figure 5.5: Results of the proposed multilevel multirate partitioning compared to unirate partitioning. (a) Relative cut quality: average cut size improvement from 20 pairs of runs, (b) relative cut quality: minimum cut size in 20 runs, (c) relative run time.

unclustering.

5.4 Conclusion

In this chapter, we deeply analyzed limitations of the multilevel partitioning paradigm and suggested a simple yet effective approach to providing flexibility to the multilevel partitioning based on multirate unclustering. An improved version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree constructed during the coarsening phase has been introduced; the multilevel uncoarsening phase involves multi-rate unclustering, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels. The preliminary experimental results show that the proposed approach is promising in providing a reasonable trade-off between cut quality and run time.

Chapter 6

Conclusion

In this dissertation, the limitations of current partitioning techniques are identified, and new multilevel partitioning algorithms have been suggested. Some new metrics for additional qualities of partitioning solutions were defined, and novel partitioning techniques on the multilevel paradigm were presented.

A new multilevel partitioning framework that takes advantage of user design hierarchy has been presented in Chapter 2. As a guidance of design hierarchy, clustering scope restriction is used to construct a multilevel clustering tree. The clustering scopes are selectively determined by Rent exponent computation and updated dynamically while the clustering tree is being built up. Due to the benefit from the guidance by the design hierarchy which has implications on connectivity between functional blocks, our proposed algorithm generates better multilevel clustering tree while the number of levels is aggressively reduced. Our experiments on large scale real circuits show that the proposed partitioning approach outperforms the state-of-the-art multilevel partitioner hMetis, producing much more stable so-

lutions.

In Chapter 3, we proposed a new incremental partitioning algorithm. Stability is defined as an additional quality of a partitioning solution, and this meta-data is formulated as a quality factor and incorporated with cut quality to constitute a multi-objective cost function. For a partially modified netlist hypergraph and a previous partitioning solution of the original netlist, the proposed algorithm produces a similar partition to the previous partitioning solution while the cut quality of the resulting partition is superior (or comparable) to fresh multiple runs of the state-of-the-art partitioner, hMetis. The trade-off between similarity and cut quality with respect to a varying similarity coefficient is observed in the experimental results. Furthermore, the algorithm has been incorporated into the multilevel paradigm to produce the comparable results in enormously reduced run times. The proposed algorithm is beneficial to ECO applications, since our approach helps block-level ECO placers maximize the incremental capability by minimizing the portions to be re-placed.

In Chapter 4, we presented a multilevel partitioning for uniform resource utilization. Based on our resource utilization model, a new quality metric of a partitioning solution, crowdedness, was defined as the weighted sum of logic occupation and the local connectivity. Using the crowdedness metric, we explored the new partitioning solution space where the local interconnections are adaptively adjusted according to the block sizes, still under the same objective of overall interconnections minimization. By the carefully designed prioritized cell move policy, the proposed crowdedness-based partitioning achieves near-optimal solutions in terms of resource utilization distribution, while the overall interconnection quality also is

improved but the feasibility is barely violated. The proposed approach is practically beneficial to multi-FPGA applications, in which excessive interconnections for a FPGA generate additional logics inside of the FPGA.

Finally, in Chapter 5, we deeply analyzed the limitation of the multilevel partitioning paradigm and suggested a simple yet effective approach to providing flexibility to the multilevel partitioning based on multirate unclustering. An improved version of the multilevel partitioning which utilizes the cluster quality statistics from the multilevel clustering tree constructed during the coarsening phase has been introduced; the multilevel uncoarsening phase involves multi-rate unclustering, where some ill-formed intermediate clusters are identified earlier and unclustered at faster rates across the levels.

Bibliography

[1] C. Alpert. "The ISPD98 Circuit Benchmark Suite," in *Proc. ACM International Symposium on Physical Design*, pp. 18–25, 1998.

[2] C. Alpert, J. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Trans. Computer-Aided Design*, vol. 17, no. 8, pp. 655–667, 1998.

[3] C. J. Alpert and A. B. Kahng, "A general framework for vertex orderings, with applications to netlist clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 63–67, 1994.

[4] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration, the VLSI Journal*, pp. 1–81, 1995.

[5] C. J. Alpert and S. Z. Yao, "Spectral partitioning, the more eigenvectors the better," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 195–200, 1995.

[6] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley Pub. Co., 1990.

[7] D. Behrens, K. Harbich, and E. Barke, "Hierarchical partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 470–477, 1996.

[8] C. Berge, *Graphs and Hypergraphs*, American Elsevier, 1976.

[9] M. A. Breuer, "A Class of Min-Cut Placement Algorithms," *Journal of Design Automation and Fault Tolerant Computing*, vol. 1, no. 4, pp. 343-382, 1977.

[10] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Pub., 1992.

[11] A. D. Caldwell, A. B. Kahng, and I. L. Markov, "Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning," *ACM Journal on Experimental Algorithms*, vol. 5, 2000. [Online]. Available: http://www.jea.acm.org/2000/CaldwellDesign/

[12] A. D. Caldwell, A. B. Kahng, and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 477–482, 2000.

[13] A. D. Caldwell, A. B. Kahng, and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning," in *Proc. ACM/IEEE Asia Pacific Design Automation Conf.*, pp. 661–666, 2000.

[14] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph Partitioning for VLSI CAD: Methodology for Heuristic Development, Experimentation and Reporting," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 349–354, 1999.

[15] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph Partitioning With Fixed Vertices," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 355–359, 1999.

[16] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "Spectral K-Way Ratio-Cut Partitioning and Clustering," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 749–754, 1993.

[17] Yongseok Cheon and D. F. Wong, "Stable Circuit Partitioning for ECO," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* under review.

[18] Yongseok Cheon and D. F. Wong, "Crowdedness-Balanced Multilevel Partitioning for Uniform Resource Utilization," in *Proc. ACM/IEEE Asia Pacific Design Automation Conf.*, under review.

[19] Yongseok Cheon, S. Lee and D. F. Wong, "Stable Multiway Circuit Partitioning for ECO," in *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp. 718–725, 2003.

[20] Yongseok Cheon and D. F. Wong, "Design Hierarchy Guided Multilevel Circuit Partitioning," in *Proc. ACM Int'l Symposium on Physical Design*, pp. 30–35, 2002.

[21] Yongseok Cheon and D. F. Wong, "Design Hierarchy-Guided Multilevel Circuit Partitioning," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 22, no.4, pp. 420–427, Apr. 2003.

[22] N. Chou, L. Liu, C. Cheng, W. Dai, and R. Lindelof, "Local ratio cut and set covering partitioning for huge logic emulation system," *IEEE Trans. Computer-Aided Design*, vol. 14, no. 9, pp. 1085–1092, 1995.

[23] C.-S. Choy, T.-S. Cheung, and K.-K. Wong, "Incremental layout placement

modification algorithms," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 4, pp. 437–445, 1996.

[24] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu, "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 441–446, 1997.

[25] J. Cong and M. Sarrafzadeh, "Incremental physical design," in *Proc. ACM International Symposium on Physical Design*, pp. 84–92, 2000.

[26] J. Cong and M'L. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 755–760, 1993.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 2nd ed.*, McGraw-Hill, 2001.

[28] J. Crenshaw, M. Sarrafzadeh, P. Banerjee, and P. Prabhakaran, "An incremental floorplanner," in *Proc. Great Lakes Symposium on VLSI*, 1999.

[29] A. E. Dunlop and B.W. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 4, no. 1, pp. 92–98, 1985.

[30] S. Dutt and W. Deng, "A probability-based approach to VLSI circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 100–105, 1996.

[31] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 194–200, 1996.

[32] W. Fang and A. C.-H. Wu, "Multi-way FPGA partitioning by fully exploiting design hierarchy," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 1, pp. 34–50, 2000.

[33] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 175–181, 1982.

[34] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[35] L. Hagen and A. B. Kahng, "A new approach to effective circuit clustering," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 422–427, 1992.

[36] L. Hagen, A. B. Kahng, F. J. Kurdahi, and C. Ramachandran, "On the intrinsic Rent parameter and spectra-based partitioning methodologies," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 1, pp. 27–37, 1994.

[37] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation part I, graph partitioning," *Operations Research*, 37, pp. 865–892, 1989.

[38] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "Gordian: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. on Computer-Aided Design*, pp. 356–365, 1991.

[39] G. Karypis, R. Aggarwal, V. Kumar, and S. Sheckhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 526–529, 1997.

[40] G. Karypis, R. Aggarwal, V. Kumar, and S. Sheckhar, "Multilevel hypergraph partitioning: Application in VLSI domain," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[41] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 343–348, 1999.

[42] G. Karypis and V. Kumar, "hMetis: A Hypergraph Partitioning Package Version 1.5.3," User Manual, November 22, 1998.

[43] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning of electrical circuits," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[44] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C. Koh, and P. H. Madden, "Recursive Bisection Based Mixed Block Placement," in *Proc. ACM International Symposium on Physical Design*, pp. 84–89, 2004.

[45] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 220, pp. 671–680, 1983.

[46] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Computers*, vol. 33, no. 5, pp. 438–446, 1984.

[47] H. Krupnova, A. Abbara, and G. Saucier, "A hierarchy-driven FPGA partitioning method," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 522–525, 1997.

[48] H. Krupnova, C. Rabedaoro, and G. Saucier, "FPGA partitioning for rapid

prototyping : A 1 million gate design case study," in *Proc. IEEE Int'l Workshop on Rapid System Prototyping*, pp. 128–133, 1999.

[49] R. Kuznar and F. Brglez, "PROP: A recursive paradigm for area-efficient and performance oriented partitioning of large FPGA netlists," in *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp. 644–649, 1995.

[50] S. Lee, Yongseok Cheon and D. F. Wong, "A Min-Cost Flow Based Detailed Router for FPGAs," in *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp. 388–393, 2003.

[51] Z. Li, W. Wu, X. Hong, and J. Gu, "Incremental placement algorithm for standard-cell layout," in *Proc. International Symposium on Circuit and Systems*, pp. 752–759, 2002.

[52] C. Liang and C. Ho, "A new optimization driven clustering algorithm for large circuits," in *Proc. European Design Automation Conf.*, pp. 28–32, 1993.

[53] H. Liu and D. F. Wong, "Network-flow-based multiway partitioning with area and pin constraints," *IEEE Trans. Computer-Aided Design*, vol. 17, no. 1, pp. 50–59, 1998.

[54] T. K. Ng, J. Oldfield, and V. Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 470–473, 1987.

[55] C.-I. Park and Y.-B. Park, "An efficient algorithm for VLSI network partitioning problem using a cost function with balancing factor," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 11, pp. 1686–1694, 1993.

[56] Y. Saab, "A fast and robust network bisection algorithm," *IEEE Trans. Computers*, vol. 44, no. 7, pp. 903–913, 1995.

[57] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62–81, 1989.

[58] M. Sarrafzadeh and C. K. Wong, *An Introduction to VLSI Physical Design*, McGraw Hill, 1996.

[59] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 57–62, 1972.

[60] N. Selvakkumaran and G. Karypis, "Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 726–733, 2003.

[61] D. P. Singh and S. D. Brown, "Incremental placement for layout-driven optimizations on FPGAs," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 752–759, 2002.

[62] N. Togawa, K. Hagi, M. Yanagisawa, and T. Ohtsuki, "An incremental placement and global routing algorithm for field-programmable gate arrays," in *Proc. ACM/IEEE Asia Pacific Design Automation Conf.*, pp. 519–526, 1998.

[63] P. Villarrubia. "Invited talk: Important Placement Considerations for Modern VLSI Chips," in *Proc. ACM International Symposium on Physical Design*, 2003.

[64] X. Yang, B. Choi and M. Sarrafzadeh, "Routability driven white space allocation for fixed-die standard-cell placement," in *Proc. ACM Int'l Symposium on Physical Design*, pp. 42–47, 2002.

[65] H. Yang and D. F. Wong, "Efficient network flow based min-cut balanced partitioning," in *Proc. IEEE Int'l. Conf. on Computer-Aided Design*, pp. 50–55, 1994.

[66] J. Yih and P. Mazumder, "A Neural Network Design for Circuit Partitioning," *IEEE Trans. on Computer-Aided Design*, vol. 9, no. 12, pp. 1265–1271, 1990.

[67] R. Zhou, J. Tong, and P. Tang, "FPART: A multi-way FPGA partitioning procedure based on the improved FM algorithm," in *Proc. Asia Pacific Design Automation Conf.*, pp. 513–518, 1998.

[68] "IC placement integrates with other design tasks," *EE Times*, April, 2003. [Online]. Available: http://www.eetimes.com/story/OEG20030408S0034

Vita

Yongseok Cheon was born in Seoul, Korea on June 21, 1971, the son of Dong-Young Cheon and Jung-Ae Park. He received the B.S. and M.S. degrees in Electronics and Computer Engineering from Korea University, Seoul, Korea, in 1993 and 1995, respectively. He worked toward the Ph.D. degree in Korea University and became a Ph.D. candidate in 1998. In January 1999, he entered the Graduate School of The University of Texas at Austin. Prior to his Ph.D. study at The University of Texas, he was an assistant professor at Kyung-In Women's College, Inchon, Korea, in 1998. He worked as a summer intern at Axis Systems, Sunnyvale, California, in 2000 and 2001. He published one journal paper and six conference papers, and one journal paper and one conference paper are under review for publication as of July 2004. He was awarded the first place prize for the best paper in the Samsung Paper Contest, 2000. His publications during his graduate study primarily focused on the area of computer-aided design algorithms for VLSI systems.

Permanent Address: 3368 Lake Austin Blvd., Apt. E

Austin, TX 78703

This dissertation was typeset with LaTeX 2$_\varepsilon$[1] by the author.

[1] LaTeX 2$_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.